An Optimal Resource Binding Algorithm with Inter-Transition Switching Activities for Low Power

Deming Chen and Scott Cromar

Abstract *— Resource binding, a key step encountered in behavioral synthesis, has been studied intensively in the past. Among the published results, resource binding to reduce switching activity (SA) of the design for minimizing dynamic power has been one of the actively-pursued topics. Two types of SAs can be minimized: the intra−transition SA (occurring during the propagation of a single input vector) and the inter−transition SA (occurring between different input vectors). Previous work either ignored the inter-transition SA or provided heuristic to deal with it. When the inter-transition SA was considered, it was not clear previously whether the problem could still be solved optimally. In this paper, for the first time, we demonstrate that resource binding considering inter-transition SAs can be solved in polynomial time for designs that can be represented by data-flow graphs (DFG). This is realized by transforming the problem into finding the shortest path problem in a k-dimensional graph. We also propose an efficient heuristic that uses a network-flow algorithm followed by a legalization step using a bipartite matching algorithm. Experimental results show that a considerable amount of SA reduction can be obtained compared to the previous state-of-the-art results.*

Keywords — Behavioral synthesis, resource binding, switching activity estimation, low power design

1 INTRODUCTION

Power reduction is one of the most important optimization targets for modern VLSI design. Power can be optimized across different design stages, from the system-level all the way down to the gate and transistor level. The higher the design level is, the more critical the design decisions are for the quality of the final product. It is reported that system and behavior level power optimization techniques can achieve more than 40% of power reduction [20]. In this study, we focus on behavioral synthesis for optimizing the dynamic power, which is still the dominating power source for many modern VLSI circuits.

The basic problem of behavioral synthesis or high-level synthesis is the mapping of a behavioral description of a circuit into a cycle-accurate RTL design consisting of a *datapath* and a *control unit*. A datapath is composed of three types of components: *functional units* (e.g., ALUs, multipliers, and shifters), *storage units* (e.g., registers and memory), and *interconnection units* (e.g., buses and multiplexers). The control unit is specified as a finite state machine which controls the set of operations for the datapath to perform during every control step (clock cycle). The behavioral synthesis process mainly consists of three tasks: *scheduling*, *allocation*, and *binding* (or *module assignment*). Scheduling determines when a computational operation will be executed; allocation determines how many instances of resources (functional units, registers, or interconnection units) are needed; binding binds operations, variables, or data transfers to these resources. Binding can also include module selection, which determines the types of the resources to be used (e.g., a carry look-ahead adder or a carry-save adder). Behavioral synthesis is a well studied problem [1][7][8][11][22]. In general, it has been shown that the code density and simulation time can be improved by 10X and 100X, respectively, when moved to the behavior-level synthesis from RTL synthesis [26]. Such an improvement in efficiency is much needed for design in the deep submicron era.

There are two sources of power consumption: *dynamic power* and *static power*. Dynamic power is consumed when signal transitions take place at gate outputs. Static power (also called leakage power) is consumed when the circuit is either active or idle. Dynamic power consumption is calculated as $P_d = 0.5 \times S \times C \times V_{dd}^{\ 2} \times f$, where S denotes the switching activity of the circuit, C denotes the effective capacitance, V_{dd} is the supply voltage, and f is the operating frequency. To lower dynamic power, each of these factors can be reduced. For example, using a smaller amount of resources would effectively reduce the C value in the formula (e.g., behavioral synthesis using variable precision arithmetic units [10]). Some work has utilized variable supply voltages to reduce power (e.g., [5][14][23]), where the basic idea is that operations on the non-critical paths can be driven by a lower supply voltage to save power without suffering an overall performance penalty. In this study, we will focus on reducing the switching activity (SA) for power minimization.

There are many research results that have addressed the problem of minimizing SA through behavioral synthesis. We introduce some representative work next. To address the interactions among the different tasks in behavioral synthesis, there are works that carried out scheduling, allocation, and binding simultaneously for power minimization [3][6][21]. Most of these algorithms used iterative approaches, such as the simulated annealing algorithm or the variable depth search algorithm. The advantage of such algorithms is that they can search for a global optimal solution. The potential drawback is that there is no guarantee that such an optimal solution can be found. In addition, to reduce the runtime complexity of such algorithms, each of the tasks can only be designed using unsophisticated approaches. Other works focused on one or two tasks for optimization so that they could achieve larger gains for these individual tasks. Reference [19] performed scheduling and binding to increase the opportunity for two operations to be bound into the same functional unit (FU) consecutively if they share common operands. It also performed register binding to minimize the input switching of FUs. However, their algorithms were all heuristic in nature and had no guarantee

of optimality. The work in [15] presents effective metrics to evaluate the power dissipation of scheduled data-flow graphs (DFG)[1]. It showed that metric evaluation is much faster than performing optimal binding and iterative power improvement, thus enabling fast design space exploration.

Reference [2] worked on register binding with a scheduled DFG. It is the first algorithm that presented an optimal register binding solution for SA reduction working with a DFG. The authors formulated the problem as a minimum cost clique covering of the *compatibility* graph, and solved it using a max-cost flow algorithm. However, this algorithm only considered the *intra-transition* SA, and did not consider the *inter-transition* SA (more details in Section 2). The work in [5] took one step further from [2] and addressed the binding task under multiple supply voltages for DFGs. It developed an optimal algorithm for assigning low V_{dd} to as many operations as possible, under the resource and time constraint, while at the same time minimizing the total SA. This work did not consider the inter-transition SA either. Reference [17] addressed the importance of inter-transition SA and derived a two-step approach to solve the problem. It first used a network flow algorithm to get an initial binding solution, and then it used a post-processing greedy iterative algorithm to make its solution legal.

Recently, there are works that addressed other issues during behavioral synthesis. For example, in [13], multi-cycle interconnect communication is considered during behavior synthesis to reduce system latency. Reference [26] proposed a module selection algorithm that combined design-time optimization with post-silicon tuning to maximize performance and power yield with consideration of process variation. The work [18] considered thermal optimization during resource allocation and binding to reduce the hot spots and cooling cost of the design. In [12], resource binding for effective soft error tolerance in FPGAs was studied for higher chip reliability. Reference [25] studied

[1] A DFG is a behavioral representation of a circuit that does not contain cycles and branches. DFGs represent data-intensive circuits commonly encountered in DSP applications and other arithmetic-centric applications.

behavioral synthesis for digital microfluidic biochips. It targeted next-generation system-on-chip (SOC) designs that are expected to include microfluidic components.

In this study, we present an algorithm that shows that resource binding considering inter-transition SAs for DFGs can be solved optimally. We achieve such a goal by transforming the problem into the shortest path problem in a *k-dimensional graph* extended from the algorithm in [16]. We derive the complexity of the algorithm and show that it can be solved in polynomial time. The order (or degree) of the polynomial, k, is the number of allocated resources. In general, k is a constant, which is specified by designers to fulfill area/power constraints for the design. The runtime of the algorithm is reasonable when k is small. When k is large, the algorithm's complexity becomes high, especially for large benchmarks. To deal with large k values, we also propose a heuristic that uses a network-flow algorithm, followed by a legalization step as done in [17]. However, our enhanced legalization step differs from [17] in that it uses a bipartite matching algorithm. Experimental results show that this legalization heuristic produces better results than the heuristic in [17] and can obtain the optimal solution for several benchmarks.

The remainder of this paper is organized as follows. Section 2 provides some key definitions and the problem formulation. Section 3 presents both the new heuristic and the optimal algorithm. Section 4 shows experimental results, and Section 5 concludes this paper.

2 DEFINITIONS AND PROBLEM FORMULATION

In a DFG $G = (V, A)$, set V corresponds to operations and set A corresponds to data transfers between operations. An edge $a = (x, y) \mid x, y \in V, a \in A$ indicates there is a data dependency between operations x and y. Scheduling assigns operations to control steps so that the overall execution latency meets a certain time constraint, and the number of resources used also meets a certain resource constraint. After scheduling, the lifetime of each operation in the DFG is the time during

which the operation is active. A *compatibility graph* $G_c = (V_c, A_c)$ for these operations can then be constructed for addition and multiplication (and other types if any) separately. V_c corresponds to all the operations of the same type, and there is a directed edge $a_c = (v_i, v_j) \mid a_c \in A_c$ between two vertices if and only if their corresponding lifetimes do not overlap, and operation v_i comes before v_j. In such a case, we call operations v_i and v_j *compatible* with each other, and they can be bound into a single FU without lifetime conflicts. Let w_{ij} denote the weight of edge a_c, which represents the cost when we bind v_i and v_j into the same FU. This cost is the switching activity between these two operations when v_j executes after v_i on the FU. If there are idle cycles between the execution of v_j and v_i, we assume that the inputs of the FU remain the same during the idle cycles so that there is no switching activity on the FU. Compatibility graphs have a transitivity property, which indicates that if there are edges (v_x, v_y) and (v_y, v_z), then there will be an edge (v_x, v_z).

Figure 1(a) shows an example of a scheduled DFG that consists of additions. Figure 1(b) is the compatibility graph generated for 1(a). The binding problem basically becomes finding multiple disjoint paths in the compatibility graph, where each path represents one FU that binds all the nodes on that path.

Now we introduce the concepts of *intra-transition SA* and *inter-transition SA*, which were first defined in [17]. In Figure 1(b), the number of adders allocated is three, which is equal to the maximum number of operations scheduled in a single control step (cstep3 in the example). Suppose the binding solution is *path1 = {op1, op3, op5}* $\Rightarrow$ *adder1*; *path2 = {op2, op6, op8}* $\Rightarrow$ *adder2*; and *path3 = {op4, op7}* $\Rightarrow$ *adder3*. If input vector PI^1 arrives at the primary inputs, after four cycles, all the corresponding outputs for the design are computed. Consecutively, input vector PI^2 can arrive and go through the same propagation. Take *adder1* for example. There is switching on the adder's ports when execution switches from *op1* to *op3* and then to *op5* when PI^1 propagates through the design. The SA incurred in this iteration is called the intra-transition SA. When PI^2 arrives, the execution would actually switch from *op5* back to *op1* to execute the new vector. The SA incurred across such iteration boundaries is called

the inter-transition SA. Figure 1(b) can model the intra-transition SA by assigning a weight on an edge which represents the SA between the two operations. However, it cannot model the inter-transition SA.

To model the inter-transition SA, the compatibility graph can be manipulated through rotation and duplication introduced in [17]. Figure 1(c) shows the manipulated compatibility graph. The line section *b* in Figure 1(b), which contains the control step with max allocation, is moved to the front of the graph and the operations in the max-allocation control step are duplicated into nodes 5', 6', and 7'. Moving the control step with the max-allocation to the front does not change the property of the compatibility graph, but will make the binding algorithms feasible (more details later). This rotation and duplication procedure is generic and can handle any compatibility graph. We call the new graph a *transformed compatibility graph* $G^T = (V^T, A^T)$. The solid lines represent the original intra-transition edges and the dashed lines represent the newly added inter-transition edges. (The inter-transition edges from node 8 to nodes 3, 4, 5', 6', 7' are not drawn to have a clearer illustration.) These edges are not real compatibility edges, but are added to provide a means for considering the inter-transition SA. Thus, the *transformed compatibility graph* is able to capture all the intra- and inter-transition edges. By adding SA values as weights on these edges, binding to reduce both intra- and inter-transition SA would become possible. For example, now, a path {5, 1, 5'} would represent that *op1* and *op5* are bound into one FU, and such a decision is reached by considering both the inter-transition SA of edge (5, 1) and the intra-transition SA of edge (1, 5'). However, to have a legal solution, every binding path needs to start from one of the nodes *v,* from the first column of the transformed compatibility graph, and end by the correspondingly duplicated node *v'* on the last column. We call this constraint the *matching node constraint*.

Our problem can be simply formulated as follows:

Problem: Low-power binding with inter-transition SA. We name this problem the **IT-SA-Bind** problem.

Given: (1) A scheduled DFG $G(V, A)$; (2) A set of available functional units R; (3) SA on the intra- and inter-transition edges.

Objective: Bind all operations to functional units under the resource-constraints so that the total SAs of functional units are minimized.

3 ALGORITHM DESCRIPTION

We first introduce our SA estimation method in Section 3.1. We then present a heuristic algorithm in Section 3.2 that offers a new legalization step over what was used in [17] to solve the **IT-SA-Bind** problem. Finally, we present the optimal algorithm to solve the same problem in Section 3.3.

3.1 SA (Switching Activity) Estimation

Our algorithm begins by carrying out a DFG simulation for SA estimation. This is similar to the approach presented in [3]. The difference is that we focus on the intra- and inter-transition SA estimations between any two compatible operations instead of targeting an operation set as done in [3].

We introduce some related definitions next. For two operations x and y, we define $C_{intra}(x, y)$ as the *toggle count* between x and y when the FU switches the execution from x to y in the same iteration (intra-transition). Similarly, we define $C_{inter}(x, y)$ as the *toggle count* between x and y when the FU switches the execution from x to y across two different iterations (inter-transition). Let $(PI^1, PI^2, \ldots, PI^K)$ be a sequence of vectors enforced on the primary inputs of the DFG G. By performing functional simulation on G, with primary input vector PI^j $(1 \le j \le K)$, we can obtain input bit vector I_i^j for operation i $(1 \le i \le |V|)$. I_i^j is computed based on the propagation of PI^j through the design when the propagation reaches operation i. We have:

$$C_{\text{intra}}(x,y) = \sum_{j=1}^{K} D_H(I_x^j, I_y^j) \qquad (1)$$

$$C_{\text{inter}}(x,y) = \sum_{j=1}^{K-1} D_H(I_x^j, I_y^{j+1}) \qquad (2)$$

where $D_H(P, Q)$ represents the Hamming Distance between bit vectors P and Q. We use an example next to illustrate how equations (1) and (2) are defined.

A simple DFG is given in Figure 2. At each control step (clock cycle), the primary inputs *a* and *b* will take one set of input vectors. For example, at cycle one, the input vectors $PI^1 = \{0001, 0101\}$ as shown in the figure. These vectors will be operated upon, and their effects will propagate through the whole DFG and we have $I_2^1 = \{0110, 0011\}$ and $I_3^1 = \{1001, 1100\}$ as marked in the figure. Note that, in the example, values (0011) and (1100) for operations *2* and *3* are constants. At cycle two, another set of primary-input vectors arrive $PI^2 = \{0111, 0000\}$. Then, $I_2^2 = \{0111, 0011\}$ and $I_3^2 = \{1010, 1100\}$. Then, using operations 2 and 3 in the example, $C_{intra}(2, 3) = D_H(I_2^1, I_3^1) + D_H(I_2^2, I_3^2) = 8 + 7 = 15$; and $C_{inter}(3, 2) = D_H(I_3^1, I_2^2) = 7$. Similar toggle count computations can be carried out for any two compatible operations to estimate the switching activity between these two operations if they are bound together into the same FU.

Switching Activity S_{intra} and S_{inter} are computed as follows:

$$S_{\text{intra}}(x,y) = \frac{C_{\text{intra}}(x,y)}{2 \times BitWidth \times K} \qquad (3)$$

$$S_{\text{inter}}(x,y) = \frac{C_{\text{inter}}(x,y)}{2 \times BitWidth \times (K-1)} \qquad (4)$$

where *BitWidth* is the bitwidth value of the input port of the FU. Note that in the formula and the example, we only considered the two input ports of an FU. We can perform similar computation for the output port of the FU. The final SA should count both the input-port SA and the output-port SA.

3.2 A Heuristic Solution with Network Flow and Bipartite Matching Algorithms

In [17], the authors added a super source and a super sink into the transformed compatibility graph to generate a network flow graph and solved a min-cost max-flow network problem to reduce the SA. Since the control step of the max-allocation is moved to the front (Figure 1(c)), the generation of the flow graph is easier. All the nodes and edges from the transformed compatibility graph are kept in the flow graph, and then the super source node is connected to all the nodes in the front control step and the super sink node is connected to all the nodes in the last column (the duplicated column). Then, k single-capacity flows with a minimum total cost can start from the super source and end at the super sink, where k is the number of nodes in the front column, i.e., the one with the max allocation. This solution will generate k disjoint paths, where each path can be bound into one resource. Since there is no way to enforce the matching node constraint over the network flow solution, the solution may not be legal. For example, it may generate a solution where {6, 8, 2, 4, 7'} in Figure 1(c) are on a single path to be bound together. This is illegal because it does not correspond to a practical binding solution. The authors in [17] then used a greedy algorithm to make the solution legal. It first came up with a conflict graph that reflects the conflicting relation of the paths that need to be legalized. It then started legalizing one path at a time iteratively based on the increasing order of the legalization cost of the paths until all the paths are legalized.

We present a new heuristic that would improve the legalization step to solve the **IT-SA-Bind** problem. We generate an initial solution using the same min-cost max-flow network method as used in [17]. We then formulate the legalization problem into a bipartite matching problem, and legalize all the paths together. Figure 3 demonstrates this idea. Figure 3(a) shows four disjoint paths

generated by the network flow solution, where the top three paths are illegal. To legalize the path starting with *a* (named as *path a*), we can either switch edge (*a, e*) to (*a, f*) $\Rightarrow$ {*a, f, g, a'*}, marked as fix front; or switch edge (*k, c'*) to (*k, a'*) $\Rightarrow$ {*a, e, k, a'*}, marked as fix back. Similarly, we can derive the fix scenarios for other paths. As a result, we can build a bipartite graph as shown in Figure 3(b), where the two selected columns from Figure 3(a) become the two disjoint sets of the bipartite graph. The costs on the edges are shown in the figure. For example, bipartite edge (*a, g*) represents the fix-front scenario and its cost is *s(a, f) + s(g, a')*, where *s(a, f)* represents the switching activity of edge (*a, f*) in Figure 3(a). Similarly, the weight for bipartite edge (*a, k*) is *s(a, e) + s(k, a').* This provides a way to evaluate which fix scenario would provide a better legalization for *path a*. However, such a local decision can have a global impact. For example, if we pick (*a, k*), then (*c, k*) cannot be used to legalize the *path c*. Therefore, we should obtain a minimum weight matching for the graph and achieve a global low cost to legalize all the paths. Figure 3(c) shows such a matching solution. Figure 3(d) shows the corresponding legalization solution. Notice that the legalization solution actually depends on which two columns are selected from Figure 3(a) in the first place to build the bipartite graph. To achieve a global minimization, we repeat this procedure for all combinations of columns, and the legalization with the lowest weight is chosen as the final solution. Although in the example all columns have the same number of nodes, this algorithm is general and can work for any transformed compatibility graphs. The complexity of the algorithm is $O(L^2 \cdot |R| \cdot (|R| \log |R|))$, where L is the schedule length for the DFG, and R is the set of FUs.

3.3 Optimal Binding with k-Dimensional Graph

An obvious attempt to derive an optimal solution for the **IT-SA-Bind** problem is to use the multi-commodity network flow algorithm with multiple sources and sinks. However, this solution has, in general, an exponential complexity and is therefore not interesting to us. Instead, we approach this problem through another route: transforming the original problem into finding the shortest path in a *k*-

dimensional graph. The *k*-dimensional graph was introduced in [16] to find *k* disjoint paths in a DAG (directed acyclic graph) between a single source *s* and a single sink *t* such that the total cost of the paths was minimized. We borrow this concept, extend it to the multi-source/multi-sink scenario based on the transformed compatibility graph, and use it to obtain an optimal legal binding solution. A legal solution would honor the matching node constraint, while at the same time ensure that all the nodes are bound using *k* resources, i.e., that the disjoint paths cover all the nodes in the graph (we name this the *node coverage constraint*).

We are given a transformed compatibility graph (which is a DAG) $G^T = (V^T, A^T)$ with sources $\{s_1, s_2, \ldots, s_k\}$ and sinks $\{t_1, t_2, \ldots, t_k\}$ as described in Section 2. There is cost on each edge (u, v) in A^T. We are interested in finding *k* vertex-disjoint paths in G^T, $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots, s_k \rightarrow t_k$, with the minimum total edge cost, which corresponds to the maximal SA reduction. Since G^T is acyclic, we can number the vertices in a topological order, i.e., if there is an edge (u, v) in G^T, then the order number assigned for *u* must be smaller than that for *v* (denoted $u < v$). We assign the orders as $s_1 = 1, s_2 = 2, \ldots, s_k = k$; and we assign $t_1 = |V^T| - k + 1, t_2 = |V^T| - k + 2, \ldots, t_k = |V^T|$. Define a *k*-dimensional graph $G^k = (V^k, A^k)$ as follows:

$$V^k = \{<v_1, v_2, \ldots, v_k> \mid v_1, v_2, \ldots, v_k \in V^T \text{ and } (v_i \neq v_j \text{ if } i \neq j)\}$$

$$A^k = \{a^k = (<v_1, v_2, \ldots, v_i, \ldots, v_k> \rightarrow <v_1, v_2, \ldots, v_i^*, \ldots, v_k>) \mid a^T = (v_i \rightarrow v_i^*) \in A^T \text{ and } v_i = min\{v_1, v_2, \ldots, v_k\}\}$$

where the cost of edge $(<v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k> \rightarrow <v_1, \ldots, v_{i-1}, v_i^*, v_{i+1}, \ldots, v_k>)$ in A^k is equal to the cost of edge $(v_i \rightarrow v_i^*)$ in A^T. The statement $v_i = min\{v_1, v_2, \ldots, v_k\}$ indicates that v_i has the minimum order number among the *k* vertices. Figure 4 shows an example, where Figure 4(c) is the 2-dimensional graph ($k = 2$ in this case) for the transformed compatibility graph G^T in Figure 4(b). For example, there is an edge $(<1, 2> \rightarrow <4, 2>)$ in Figure 4(c) because edge $(1 \rightarrow 4)$ is in G^T, and the

order number of 1 is less than 2. Similarly, there is an edge ($<3, 2> \rightarrow <3, 4>$) because there is an edge ($2 \rightarrow 4$) in G^T, and the order number of 2 is less than 3. Note that not all the edges are shown in Figure 4(c) to avoid a messy picture. We have the following theoretical results.

Lemma 1. There exist k vertex-disjoint paths $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots, s_k \rightarrow t_k$ in a DAG, $G^T = (V^T, A^T)$, with the minimum total edge cost if and only if there exists a directed shortest path P^k from $<s_1, s_2, \ldots, s_k>$ to $<t_1, t_2, \ldots, t_k>$ in the k-dimensional graph G^k.

Proof:

(**if**) Given k vertex-disjoint paths in G^T: $P_j = u_1^{(j)}, u_2^{(j)}, \ldots, u_{Xi}^{(j)}, j = 1, \ldots, k,$ where $u_1^{(j)} = s_j$ and $u_{xi}^{(j)} = t_j$, we arrange the edges of $P_1, P_2, \ldots, P_k$ in ascending order of their edge sources to form the sequence ($w_1 \rightarrow z_1, w_2 \rightarrow z_2, \ldots, w_L \rightarrow z_L$). Note that $u_1^{(1)} \rightarrow u_2^{(1)}, u_1^{(2)} \rightarrow u_2^{(2)}, \ldots, u_1^{(k)} \rightarrow u_2^{(k)}$ are the first k edges in the sequence. Also we have $s_1 < s_2 < \ldots < s_k < w_{k+1} < \ldots < w_L$ and $L = \sum_1^k (X_i - 1)$.

Define a path $P^k = (a_1, \ldots, a_L)$ in G^k with edges $a_1, a_2, \ldots, a_L$ as

$a_1 = <s_1, s_2, \ldots, s_k> \rightarrow <z_1, s_2, \ldots, s_k>$

$a_2 = <z_1, s_2, \ldots, s_k> \rightarrow <z_1, z_2, \ldots, s_k>$

...

$a_{k-1} = <z_1, z_2, \ldots, s_{k-1}, s_k> \rightarrow <z_1, z_2, \ldots, z_{k-1}, s_k>$

$a_k = <z_1, z_2, \ldots, z_{k-1}, s_k> \rightarrow <z_1, z_2, \ldots, z_{k-1}, z_k>$

$a_q = <n_1, n_2, \ldots, n_{i-1}, w_q, n_{i+1}, \ldots, n_k> \rightarrow <n_1, n_2, \ldots, n_{i-1}, z_q, n_{i+1}, \ldots, n_k>,$

if $w_q \rightarrow z_q$ is on one of the k paths in G^T, and $q = k+1, \ldots, L$

…

It is easy to check that $a_1, \ldots, a_L$ are well defined and that P^k is a path from $<s_1, s_2, \ldots, s_k>$ to $<t_1, t_2, \ldots, t_k>$ in G^k.

(**only if**) Suppose there exists a path $P^k = <x_1^{(1)}, x_2^{(1)}, \ldots, x_k^{(1)}> \rightarrow <x_1^{(2)}, x_2^{(2)}, \ldots, x_k^{(2)}> \rightarrow \ldots \rightarrow <x_1^{(r)}, x_2^{(r)}, \ldots, x_k^{(r)}>$ in G^k, where $<x_1^{(1)}, x_2^{(1)}, \ldots, x_k^{(1)}> = <s_1, s_2, \ldots, s_k>$ and $<x_1^{(r)}, x_2^{(r)}, \ldots, x_k^{(r)}> = <t_1, t_2, \ldots, t_k>$. The definition of G^k implies that for each $i = 1, 2, \ldots, r$, there exists $j^* \in \{1, \ldots, k\}$ such that $x_{j^*}^{(i)} \rightarrow x_{j^*}^{(i+1)} \in A^T$ and $x_j^{(i)} = x_j^{(i+1)}$ for every other $j \in \{1, \ldots, k\}$, where $j \neq j^*$. An important property is that the order of the appearance of j^* in the first k edges of P^k is $s_1, s_2, \ldots, s_k$ as shown in edges $a_1, a_2, \ldots, a_k$ above in the **if** part of the proof. This is achieved because of the node order assignment: $s_1 = 1, s_2 = 2, \ldots, s_k = k$. Meanwhile, we have the following node order assignment: $t_1 = |V^T| - k + 1, t_2 = |V^T| - k + 2, \ldots, t_k = |V^T|$. As a result, $P_j = x_j^{(1)} \rightarrow x_j^{(2)} \rightarrow \ldots \rightarrow x_j^{(r)}, j = 1, \ldots, k,$ are vertex-disjoint s_j to t_j paths in G^T, where $x_j^{(1)} = s_j$ and $x_j^{(r)} = t_j$. Note that t_j is the duplicated s_j node in G^T, i.e., $t_j = s_j$' (Section 2). Therefore, the *matching node constraint* can be fulfilled in these k vertex-disjoint s_j to t_j paths.

Furthermore, the total cost of the edges in P^k is equal to the total edge cost of the corresponding k paths in G^T. Therefore, if P^k is the shortest path from $<s_1, s_2, \ldots, s_k>$ to $<t_1, t_2, \ldots, t_k>$, the k paths in G^T are those paths with the minimum total edge cost. ■

Theorem 1. Let the edge cost of every edge in G^T be a negative value, and for any two consecutive edges (v_x, v_y) and (v_y, v_z), let $cost(v_x, v_y) + cost(v_y, v_z) < cost(v_x, v_z)$, then the shortest P^k path in G^k will produce k vertex-disjoint paths in G^T as specified in Lemma 1, with the minimum total edge cost, covering all the vertices in G^T.

Proof:

First, we show that k vertex-disjoint paths can cover all the vertices in G^T. The compatibility relation on V^T makes V^T a partially ordered set. (Please refer to [3] for the definition of partially ordered sets.) A subset of V^T, which contains the largest number of mutually non-compatible nodes, has cardinality k. Dilworth's theorem [8] indicates that a partially ordered set P can be partitioned into k-disjoint paths covering all the elements if P contains at least one subset Y, where $|Y| = k$; every pair of elements in Y are non-compatible with each other; and k is the largest number for such kind of subsets in P. These properties hold exactly in V^T.

Because the edge cost of every edge in G^T is defined as a negative value, and for any two consecutive edges (v_x, v_y) and (v_y, v_z), $cost(v_x, v_y) + cost(v_y, v_z) < cost(v_x, v_z)$, we can conclude that the more vertices that are included in the k vertex-disjoint paths in G^T, the lower the total edge cost would be. A special case occurs if a node, m, is not compatible with any other nodes in the original compatibility graph $G_c = (V_c, A_c)$. After the transformation, there will be an inter-transition edge ($m \rightarrow m'$) in G^T because m would be in the control step with the max resource allocation, where m occupies a resource just by itself. In this case, the edge ($m \rightarrow m'$) will still be included in the final solution because including this edge would lower the total cost (edge cost of every edge in G^T is negative). In conclusion, the minimum-cost solution will include all the nodes in V^T, and the *node coverage constraint* can be fulfilled. Such a min-cost solution can be found by computing the shortest P^k path in G^k. ■

Theorem 1 guarantees that we can have a legal solution which is optimal in terms of SA reduction for G^T. To fulfill the negative and inequality cost requirements in Theorem 1, we define the edge cost as $cost(x, y) = s(x, y) - (3S^{max} - S^{min})$, where $s(x, y)$ is the SA on edge (x, y), S^{max} is the maximum SA in G^T, and S^{min} is the minimum SA in G^T. Note that the smaller the positive value $s(x, y)$ the smaller the negative value $cost(x, y)$. In Figure 4(c), the shortest path P^2 is shown in bold. Note that Figure 4(c) shows symmetry for the edges. This is only because G^T in the example is symmetric. In general, the

edges in the k-dimensional graph do not need to be symmetric. The shortest path solution P^2 contains two disjoint paths for G^T. We can simply extract out the embedded G^T edges in the shortest path, which are ($1 \rightarrow 4$), ($2 \rightarrow 3$), ($3 \rightarrow 2'$), and ($4 \rightarrow 1'$). These edges form two disjoint paths: $1 \rightarrow 4 \rightarrow 1'$ and $2 \rightarrow 3 \rightarrow 2'$. Figure 4(d) shows the final binding solution from these two disjoint paths. We observe that both the matching node constraint and the node coverage constraint are met.

Theorem 2. The **IT-SA-Bind** problem can be solved optimally in polynomial time for DFG.

Proof:

Given $G^T = (V^T, A^T)$ as the transformed compatibility graph for the DFG, and k is the number of allocated resources, which is equal to the number of nodes in the first control step in G^T. Let edge $e = (a \rightarrow b) \in A^T$, define all the edges in the k-dimensional graph G^k that correspond to e. Based on the definition of G^k, these edges look like the following:

$a{*}{*}{*}{*} \rightarrow b{*}{*}{*}{*}$

$*a{*}{*}{*} \rightarrow *b{*}{*}{*}$

$**a{*}{*} \rightarrow **b{*}{*}$

…

where * can be any of the $|V^T|-2$ other nodes, and the number of *'s in each node above is $k-1$. The permutations of *'s (where order matters) is:

$$(k-1)!\binom{|V^T|-2}{k-1} \leq \left(|V^T|-2\right)^{(k-1)}$$

In addition, there are k places to put a (or b) in the node of G^k, so we have $\left(|V^T|-2\right)^{k-1} k$ new edges in G^k for each edge in G^T. Hence, the number of edges in G^k

$$\approx |A^T| \left[\left(|V^T|-2\right)^{k-1} k \right] \approx k \bullet |A^T| \bullet |V^T|^{k-1}.$$

The number of nodes in G^k is $k! \binom{|V^T|}{k} - |V^T| \leq |V^T|^k$. Shortest path in $G^k(V^k, A^k)$ can be solved in $O(|V^k|+|A^k|)$ because G^k is a DAG. Therefore, the overall complexity of the algorithm is:

$$O(k \bullet |A^T| \bullet |V^T|^{k-1} + |V^T|^k).$$

Since k is a constant that is usually specified by the designer to honor area/power constraint, this indicates that the **IT-SA-Bind** problem is solvable in polynomial time. ■

When k is small, this algorithm will use reasonable runtime as shown in the experimental results. The algorithm is especially useful for applications where a large value of resource allocation would not help improve performance anyway due to certain node dependencies (i.e., the parallelism in the application is limited). However, when k is large, the runtime of the optimal algorithm may not be affordable any more. That case, the proposed bipartite-based algorithm in Section 3.2 can be used.

Note that all the algorithms presented in this study can be easily applied to register binding and bus binding for low power, because those problems can be translated into binding with compatibility graphs as well.

4 EXPERIMENTAL RESULTS

To examine the quality gap between our optimal solution, the new heuristic, and the previous heuristic solution [17], we implemented all of these algorithms for comparison purpose. The benchmarks we use

include binary adder trees, binary multiplier trees, and some benchmarks from [24], which include several different DCT and DSP algorithms. To have a fair comparison, all the algorithms use the same scheduling result with the same resource constraint for each benchmark respectively. Scheduling is done by a list scheduling algorithm. We used a 2.8 GHz Pentium 4 machine with Linux, with 2 GB of memory.

Table I lists the results of the comparison. The second column lists the scheduling length for each benchmark. The third and fourth columns characterize the benchmarks, where *n* is the number of nodes (addition or multiplication) in the DFG, and *k* is the resource allocation for binding those nodes respectively. The next three columns list the total switching activities achieved by the three algorithms, where *Confl* is the algorithm used in [17]; *Bipar* is the new heuristic presented in Section 3.2; and *Kdim* is the optimal algorithm presented in Section 3.3. Next, columns 8 and 9 compute the percentage of SA increase for *Confl* and *Bipar* over the *Kdim* solution, using *Kdim* as the base. We can observe that *Confl* solution is on average 6.7% larger than the optimal solution (up to 20.1% individually), and *Bipar* is on average 4.1% larger than the optimal solution (up to 15.8% individually). *Bipar* is on average 2.6% better than *Confl*. Figure 5 lists the same comparison results using a bar chart. For some cases, *Bipar* achieves the optimal solution. We can also observe that in general if the ratio of *n* over *k* is larger, the gain on SA reduction is also larger. This is easy to understand because the more legalization opportunities that appear, the more optimization gain that can be obtained.

Table II lists the runtime results for these three algorithms. It is shown that the runtimes of *Confl* and *Bipar* are almost comparable. It is not surprising that *Kdim* uses the largest runtime due to its high computational complexity. However, it is still tolerable. For certain benchmarks, such as adder trees, the tradeoff between power reduction and runtime increase is significant and worthwhile.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented algorithms to solve the low-power resource binding problem considering inter-transition switching activities in the design. We proved that such a problem for DFGs can be solved optimally in polynomial time, and we presented such an optimal algorithm in detail. The optimal solution is, on average, 6.7% (up to 20.1% individually) better than a previously published algorithm. We also developed a new heuristic algorithm that, on average, is 4.1% worse when compared to the optimal solution. It is, on average, 2.6% (up to 10.5% individually) better than the previously published algorithm. In the future, we can study ways to prune the *k*-dimensional graph in terms of its nodes and edges so that it will use significantly less runtime, while still achieving near-optimal solution.

6 ACKNOWLEDGEMENT

This work was supported in part by the NSF Grant CCF 07-02501 and the NSF Career Award CCF 07-46608. We used the machines donated by Intel.

REFERENCES

[1] R. Camposano and W. Wolf. *High-level VLSI synthesis*. Springer-Verlag New York, 2001.

[2] J. M. Chang and M. Pedram. Register allocation and binding for low power. In *Design Automation Conference*, 1995.

[3] D. Chen and J. Cong. Register Binding and Port Assignment for Multiplexer Optimization. In *Asia and South Pacific Design Automation Conference,* pp. 68-73, 2004.

[4] D. Chen, J. Cong, and Y. Fan. Low-power high-level synthesis for FPGA architectures. In *International Symposium on Low Power Electronics and Design*, pages 134–139, 2003.

[5] D. Chen, J. Cong, and J. Xu. Optimal simultaneous module and multi-voltage assignment for low-power. *Trans. on Design Automation of Electronic Systems*, 11(2):362–386, April 2006.

[6] A. Dasgupta and R. Karri. Simultaneous scheduling and binding for power minimization during microarchitecture synthesis. *Int. Symp. Low-Power Electronic Design*, 1995.

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

[8] R. Dilworth. A decomposition theorem for partially ordered set. *Ann. Math 51*, pp. 161–166, 1950.

[9] J. P. Elliott. *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*. Kluwer Academic Publishers, 1999.

[10] M. Ercegovac, D. Kirovski, and M. Potkonjak. Low power behavioral synthesis optimization using multiple precision arithmetic. In *Design Automation Conference*, 1999.

[11] D. Gajski, N. Dutt, and A. Wu. *High-level synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.

[12]S. Golshan, and E. Bozorgzadeh. SEU-Aware Resource Binding for Modular Redundancy Based Designs on FPGAs. In *Intl. Conference on Design, Automation, and Test in Europe,* April, 2009.

[13]J. Jeon, D. Kim, D. Shin, K. Choi. High-level synthesis under multi-cycle interconnect delay. In *Asia and South Pacific Design Automation Conference*, 2001.

[14]M. C. Johnson and K. Roy. Datapath scheduling with multiple supply voltages and level converters. *Trans. on Design Automation of Electronic Systems*, 2(3):227–248, July 1997.

[15]E. Kursun, A. Srivastava, S. Ogrenci Memik, and M. Sarrafzadeh. Early Evaluation Techniques for Low Power Binding. In *Intl. Symposium on Low Power Electronics and Design*, Aug. 2002.

[16]C. L. Li, S. T. McCormick, and D. Simchi-Levi. Finding disjoint paths with different path-costs: Complexity and algorithms. *Networks*, 22:653–667, 1992.

[17]C.-G. Lyuh and T. Kim. High-level synthesis for low power based on network flow method. *Trans. on VLSI Systems*, 1(3):364–375, June 2003.

[18]R. Mukherjee, S. Ogrenci Memik, and G. Memik. Temperature-Aware Resource Allocation and Binding in High-Level Synthesis. In *Design Automation Conference,* June, 2005.

[19]E. Musoll and J. Cortadella. Scheduling and resource binding for low power. *Proc. Int. Symp. Syst. Synthesis*, 1995.

[20]M. Pedram. Low power design methodologies and techniques: An overview. In *http://atrak.usc.edu/~massoud/*, 1999.

[21]A. Raghunathan and N. Jha. Scalp: an iterative-improvement-based low-power data path synthesis system. *IEEE Trans. on Computer-Aided Design of ICS*, 16(11):1260–1277, Nov 1997.

[22]A. Raghunathan, N. K. Jha, and S. Dey. *High-level power analysis and optimization*. Kluwer Academic Publishers, 1998.

[23]S. Raje and M. Sarrafzadeh. Variable voltage scheduling. In *International Symposium on Low Power Design*, 1995.

[24]M. B. Srivastava and M. Potkonjak. Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput. *Trans. on VLSI Systems*, Vol. 3, No. 1, pp. 2-19, 1995.

[25]F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. *Journal on Emerging Technologies in Computing Systems*, Vol. 3, No. 4, Jan. 2008.

[26]K. Wakabayashi and T. Okamoto. C-based SoC design flow and EDA tools: an ASIC and system vendor perspective. *IEEE Trans. on Computer-Aided Design of ICS,* 19(12):1507–1522, Dec 2000.

[27]F. Wang, X. Wu, and Y. Xie. Variability-driven module selection with joint design time optimization and post-silicon tuning. In *Asia and South Pacific Design Automation Conference*, 2008.

FIGURES AND TABLES

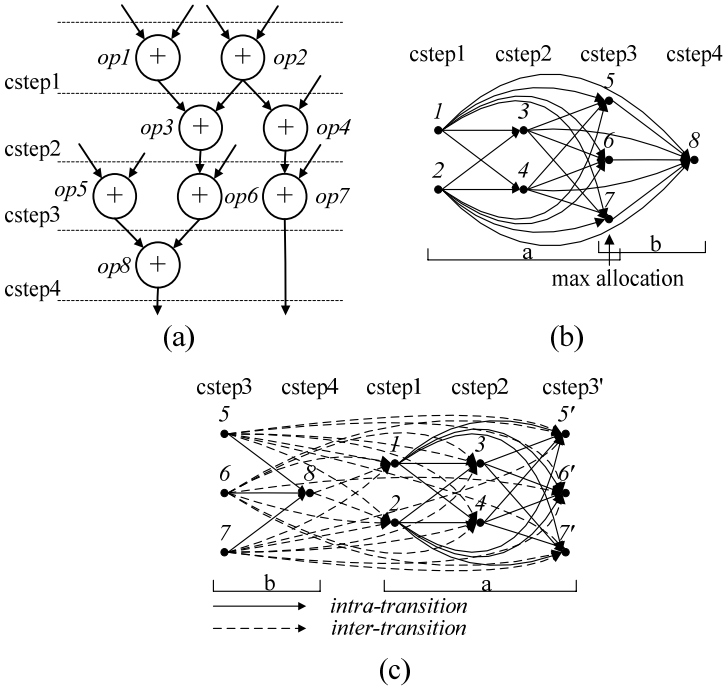

Figure 1. Compatibility graph creation and transformation. (a) Scheduled DFG. (b) Compatibility graph for the DFG in (a), with the control step of max allocation in cstep3. (c) The rotated, multi-source multi-sink compatibility graph with nodes in cstep3 duplicated.

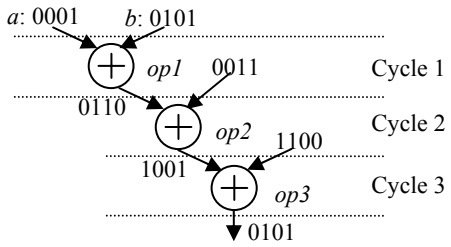

Figure 2: An example showing the toggle count computation. Only bit vectors of the first iteration are shown.

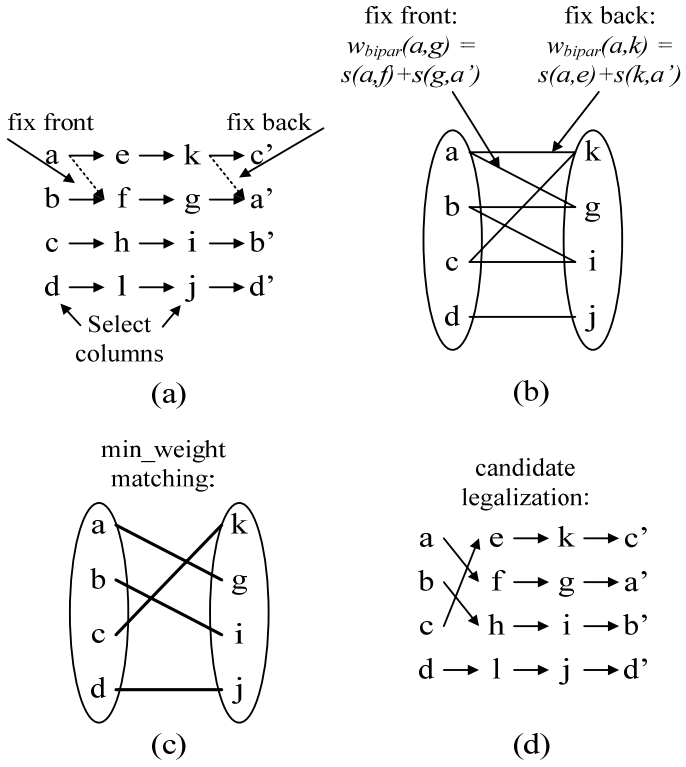

Figure 3: Legalization process. (a) An initial solution that needs to be legalized. (b) Bipartite graph construction for the two selected columns in (a). (c) The minimum weight matching solution. (d) The legalization of the paths corresponding to the matching.

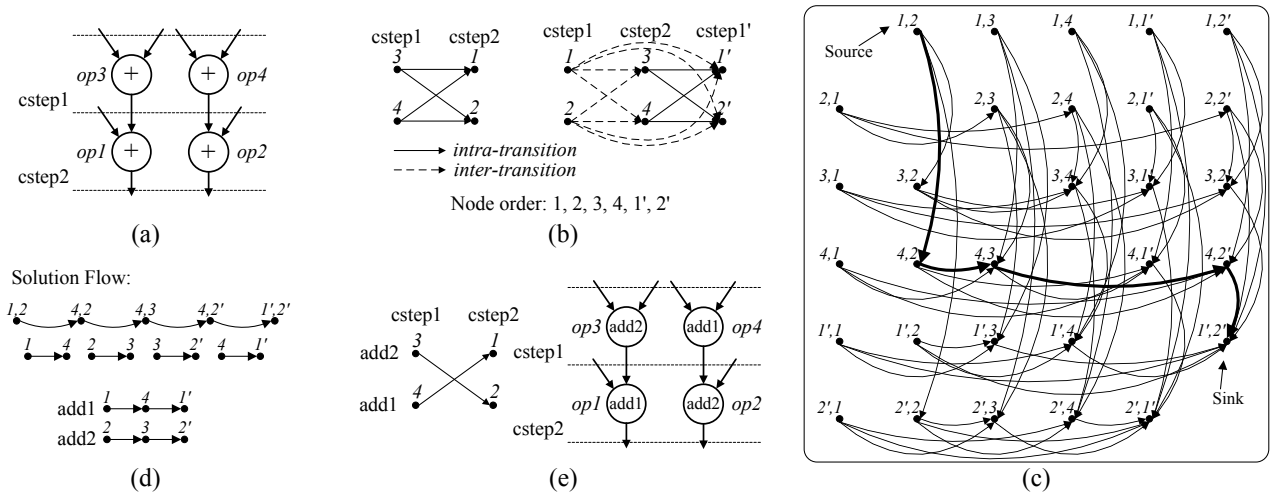

Figure 4: *K*-dimensional graph construction for binding. (a) Scheduled DFG. (b) The compatibility graph (left) and the transformed compatibility graph G^T (right) are constructed. (c) The k-dimensional graph for G^T. The shortest path is in bold. (d) From the shortest path in (c), the optimal solution of the original network is reconstructed. (e) The DFG is bound according to the optimal solution.

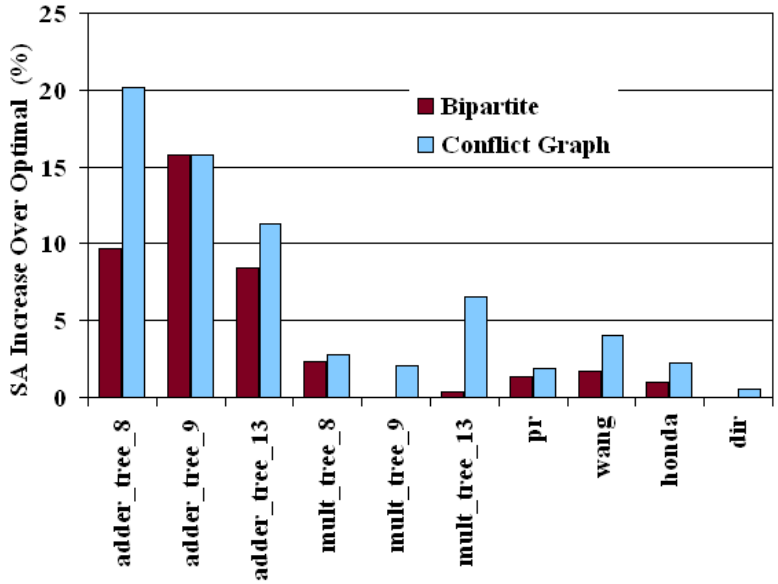

Figure 5: SA increase over the optimal solution for the *Confl* algorithm [17] and the *Bipar* algorithm.

TABLE I
COMPARISON OF THE HEURISTIC LEGALIZATION ALGORITHMS (*CONFL* [17] AND *BIPAR*) TO THE OPTIMAL SOLUTION (*KDIM*). THE LAST COLUMNS SHOW THE INCREASE IN SA OVER THE OPTIMAL SOLUTION.

Benchmarks	Sche. Length	Characterization		Total SA			SA Increase Over Kdim	
		Adds *n/k*	Mults *n/k*	Confl	Bipar	Kdim	Confl	Bipar
adder_tree_8	3	8/6		1.59	1.45	1.33	20.1%	9.6%
adder_tree_9	5	9/5		1.98	1.98	1.71	15.8%	15.8%
adder_tree_13	6	13/5		3.19	3.11	2.87	11.3%	8.4%
mult_tree_8	3		8/6	1.71	1.70	1.66	2.8%	2.4%
mult_tree_9	5		9/5	4.09	4.01	4.01	2.1%	0.0%
mult_tree_13	6		13/5	5.88	5.54	5.52	6.6%	0.4%
pr	10	26/2	16/2	12.28	12.23	12.06	1.9%	1.4%
wang	12	26/3	22/3	19.47	19.04	18.71	4.1%	1.7%
honda	23	45/3	52/3	23.04	22.76	22.53	2.3%	1.0%
dir	10	84/3	64/2	42.01	41.80	41.79	0.5%	0.0%
Average							**6.7%**	**4.1%**

TABLE II
COMPARISON OF RUNTIME OF THE ALGORITHMS

Benchmarks	Runtime (s)		
	Confl	Bipar	Kdim
adder_tree_8	0	0	406
adder_tree_9	0	0	47
adder_tree_13	0	0	376
mult_tree_8	0	0	307
mult_tree_9	0	0	46
mult_tree_13	0	0	497
pr	0	0	0
wang	0	0	16
honda	0	1	210
dir	0	4	1080

BIOGRAPHIES

Deming Chen obtained his BS in computer science from University of Pittsburgh, Pennsylvania in 1995, and his MS and PhD in computer science from University of California at Los Angeles in 2001 and 2005 respectively. He worked as a software engineer between 1995-1999 and 2001-2002. He joined the ECE department of University of Illinois, Urbana-Champaign as a faculty member in 2005. His current research interests include nano-systems design and nano-centric CAD techniques, FPGA synthesis and physical design, high-level synthesis, microprocessor architecture design under process/parameter variation, and reconfigurable computing.

He is a technical committee member for a series of conferences and symposia, including FPGA'06-10, ASPDAC'07-10, ICCD'07-09, ISQED'09-10, DAC'09-10, ICCAD'09, DATE'10, etc. He also served as session chair, panelist, panel organizer or moderator for some of these and other conferences. He is a TPC subcommittee chair for ASPDAC'09-10 and a CAD Track co-chair for ISVLSI'09 and ISCAS'10. He is an associated editor for TVLSI (IEEE Transactions on Very Large Scale Integration Systems), JCSC (Journal of Circuits, Systems and Computers), and JOLPE (Journal of Low Power Electronics). He obtained the Achievement Award for Excellent Teamwork from Aplus Design Technologies in 2001, the Arnold O. Beckman Research Award from UIUC in 2007, the NSF CAREER Award in 2008, the ASPDAC'09 Best Paper Award, and the SASP'09 Best Paper Award. He is included in the List of Teachers Ranked as Excellent in 2008.

Scott Cromar graduated from the University of California, Irvine, in 2007 with a Bachelor of Science in Electrical Engineering. He then attended the University of Illinois, Urbana-Champaign, where he completed a Master of Science in Electrical and Computer Engineering in 2009. At the University of Illinois, his research focused on robust, process variation-aware algorithms for behavioral circuit synthesis. He received the IEEE Charles LeGeyt Fortescue Graduate Scholarship

in 2007.