

Effective Techniques for the Generalized Low-Power Binding Problem

AZADEH DAVOODI and ANKUR SRIVASTAVA
University of Maryland, College Park

This article proposes two very fast graph theoretic heuristics for the low power binding problem given fixed number of resources and multiple architectures for the resources. First, the generalized low power binding problem is formulated as an Integer Linear Programming (ILP) problem that happens to be an NP-complete task to solve. Then two polynomial-time heuristics are proposed that provide a speedup of up to 13.7 with an extremely low penalty for power when compared to the optimal ILP solution for our selected benchmarks.

Categories and Subject Descriptors: B.5.2 [RTL-Implementation]: Design Aids—*Automatic synthesis, optimization*

General Terms: Algorithms, Performance, Design, Theory

Additional Key Words and Phrases: Low-power binding, graph theory, high level synthesis

1. INTRODUCTION

A portable device today is expected to perform high-speed complex tasks consuming extremely low power. Complex functionality corresponds to more transistors on a chip and consequently more power consumption. Also portability demands lower power dissipation. Power consumption is so important that it should be optimized at all the steps of the design flow.

Binding is the process of ordering the operations on available resources such that the computation could be done successfully. Low-power binding has been an active topic of interest during the past decade.

The way binding is done drastically affects the power dissipation. Among the optimization techniques at different design levels, architectural and behavioral decisions influences the design the most. Even though, so much work has been done at the behavioral level, few of them have considered the general low-power binding problem.

Given a set of operations in a scheduled DFG, the generalized low-power binding problem, binds operations to R resources and picks the architecture

Authors' address: University of Maryland, 2356 A. V. Williams Building, College Park, MD 20783, Correspondence email: azade@eng.umd.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2006 ACM 1084-4309/06/0100-0052 \$5.00

for each resource such that the overall power is minimized. This is under the assumption that different implementations or architectures of a resource type are available.

It is shown in Kruse et al. [2001] that the problem is NP-complete. Our contribution in this paper is two polynomial-time heuristics for the generalized low-power binding problem that result in very high quality solutions for our selected benchmarks.

In this article, after initially discussing relevant work in the literature, the low-power binding problem for fixed number of resources and single architecture of the resources is formulated as a min-cost flow problem that can be optimally solved [Chang and Pedram 1995].

The problem is further extended to the general case of having multiple architectures for the resources. The general binding problem is then formulated as an ILP problem that can be optimally solved but happens to be an NP-complete task [Raghunathan and Jha 1995]. Two polynomial-time heuristics are presented in this article that result in very fast high quality solutions.

The first graph-based technique iteratively runs the single-architecture flow formulation for each architecture and then chooses the least power consuming assignment from the set of resulting candidates. Afterwards, it assigns the possible unassigned operations through a node coverage algorithm that follows another flow formulation. The node coverage algorithm runs iteratively until all the unassigned operations are covered.

The second technique assigns the operations to the resources of multiple architectures in incremental clock steps similar to an approach in the left-edge algorithm [Hashimoto and Stevens 1971].

Experimental results for 12 selected benchmarks shows a speed up of up to 13.7 with a maximum penalty of 2.5% compared to the optimal ILP solution.

Section 2 discusses previous work in the literature. Section 3 formulates the low-power binding problem for the case of single and multiple architectures. Sections 4 and 5 describe the two proposed graph theoretic techniques. Experimental results are presented in Section 7 and finally conclusion is made in the last section.

2. PRELIMINARIES

The initial steps of the design flow are extremely important as they impact the final implementation significantly.

In particular, architectural decisions during High-Level Synthesis (HLS) have a profound impact on the power consumption of the final design. Therefore, power consumption is considered in high-level synthesis tools as another optimization objective.

There are three phases in the behavioral synthesis. Scheduling takes a Data Flow Graph (DFG) and assigns clock cycle to each operation. Allocation determines the number of required resources and assignment or resource binding maps the operations to the resources. Scheduling is usually done prior to the other two tasks.

Hafer and Parker [1982] can be considered as the earlier work in resource binding. There has been lots of work in literature in resource binding [Tseng

and Siewiorek 1986; Hafer and Parker 1982; Raje and Bergamaschi 1977; Fang and Wong 1994; Herrmann and Ernstl 1999; Bringmann and Rosenstiel 1997]. Tseng and Siewiorek [1986] modeled the resource-sharing problem as clique partitioning. Resource binding can be considered for functional units, registers and multiplexors. Both Hafer and Parker [1982] and Tseng and Siewiorek [1986] addressed the register and bus-sharing problem. Raje and Bergamaschi [1977] present a generalized resource-sharing methodology that interleaves register and functional unit sharing into a global clique-partitioning framework. Among binding of different resource types, functional units contribute substantially to power dissipation.

In a CMOS circuit, an important source of power dissipation is switching power which could be expressed as below [Chandrakasan et al. 1992]:

$$Power = K.V^2.\beta, \quad (1)$$

Where

K = Proportionality constant,

V = Supply voltage,

β = Switched Capacitance factor.

Switching power is directly proportional to the switching capacitance. Switching capacitance is defined as the product of the physical gate output capacitance and transition activity at the gate. Here, for a fixed supply voltage, switched capacitance is an important optimization metric.

Most of the previous work in HLS for low-power propose estimation and optimization of power consumption at algorithmic and architectural level. In Raghunathan and Jha [1995], allocation has been formulated as an Integer Linear Programming (ILP) problem. The objective function is to minimize the overall switched capacitance in the data paths. In the ILP formulation, functional units, registers and multiplexors have been considered. In Chang and Pedram [1995], external switching activity of a set of registers has been calculated following a statistical approach for single architecture of the resources. Using these values, register allocation and binding problem has been formulated as a minimum-cost clique covering of an appropriately defined compatibility graph. In Kruse et al. [2001], the importance of computing the lower and upper bounds on power consumption are considered. Low-power allocation and binding with and without constraint on the number of resources has been formulated. Based on the calculated power cost information of the functional units, efficient heuristics are proposed to measure these bounds. Here, the resources can be considered from different architectures. Allocation and binding of a scheduled DFG for fixed number of resources and architectures is an NP-complete task [Kruse et al. 2001]. Hence, polynomial-time algorithms that efficiently solve the problem of binding the operations to the resources with minimum power consumption need to get investigated. The ILP formulation in Raghunathan and Jha [1995] can be considered for different architectures but the solution is not efficient. Register binding formulation in Chang and Pedram [1995] is for single architecture only and in Kruse et al. [2001] the main concern is to calculate lower and upper bounds on power consumption of data paths.

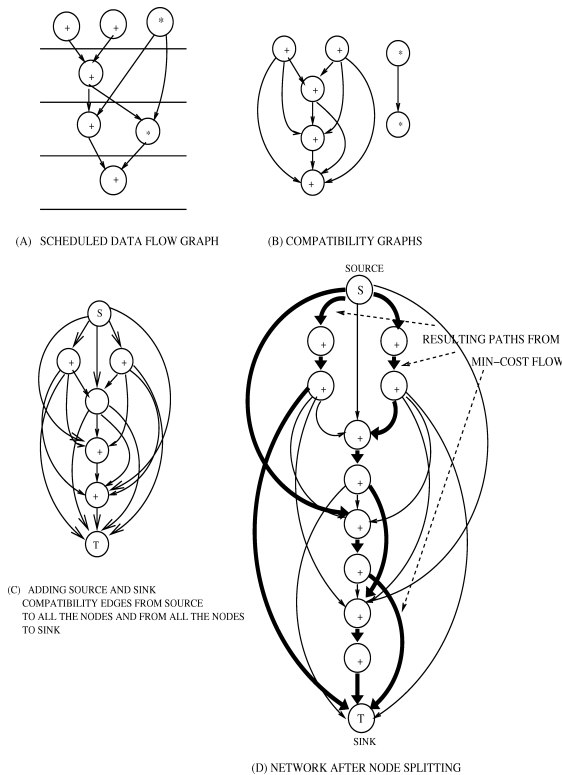


Fig. 1. Single-architecture low-power binding.

In this article, the low-power binding problem of a scheduled DFG for fixed number of resources is extended considering different architectures of the resources. Since the extended problem is NP-complete, our contribution is polynomial-time heuristics that provide great speedup up to 13.7 with very little loss of quality from the optimal ILP solution.

3. LOW-POWER BINDING PROBLEM

Given a scheduled DFG and fixed number of resources, the low-power binding problem, assigns each of the DFG operations to a resource such that the overall switched capacitance of the resources is minimized. The problem can have two variations if the architecture of the functional unit is known or not. The generalized low-power binding problem binds operations to the given R resources and picks up the architecture for each resource such that the overall power is minimized.

Assuming a single architecture for the time being for better insight, the formulation of the binding problem is described as follows based on work done in Chang and Pedram [1995].

As illustrated in Figure 1(A) and (B), the compatibility graph for a single architecture and operation type is generated from a given scheduled DFG.

Each vertex of the compatibility graph corresponds to an operation. An edge is added between any two compatible nodes. Compatible nodes are operations that can be bound on the same resource.

Given the number of resources of a certain type as the resource constraint, the problem of binding the operations of the compatibility graph onto these resources is usually modeled as a min-cost flow formulation developed in Chang and Pedram [1995] for a single architecture. The edge weights or cost values indicate the estimated switched capacitance on the given module if the source and destination of the edge form a predecessor and successor pair of operations on that resource. The cost values are calculated for the specific single architecture that is considered. Precise definition and calculation of the edge costs have been previously investigated in Kruse et al. [2001] and Raghunathan and Jha [1995]. Similar to the register binding formulation described in Chang and Pedram [1995], the node list is augmented by a sink(t) and source(s) with directed edges from the source to all the nodes and from all the nodes to the sink (Figure 1(C)). The non-sink/source nodes are further duplicated resulting in a network with vertex splitting as Figure 1(D) illustrates. Considering any duplicated pair, any incoming edge to the original node enters the top one and any outgoing edge from the original node leaves the bottom one. An edge is added between any pair of duplicated nodes. Here, the edge weight between each pair of duplicated nodes is $-W$ where W is the summation of all the edge costs. Each edge has capacitance of unity. Resource binding for a specific operation and architecture given R resources is solved by sending R units of flow from the source node to the sink such that each node is visited exactly once, all the nodes are covered and the overall cost is minimized.

Note that it is assumed that R , the number of resources of a specific type is at least the minimum number required for scheduling which is specified by the maximum number of operations scheduled on the same clock cycle. Figure 1(D) illustrates the sets of operations that are bound together. The operations on the same bold path in Figure 1(D) are assigned to the same resource.

More formally the objective and constraints can be represented as:

$$\text{Minimize } \sum_{ij \in E} X_{ij} C_{ij} \quad (2)$$

$$\sum_{\forall i \text{ such that } si \in E} X_{si} = R \quad (3)$$

$$\sum_{\forall i \text{ such that } it \in E} X_{it} = R \quad (4)$$

$$\sum_{\forall j \text{ such that } ij \in E,} X_{ij} = 1 \forall i, i \neq s, t \quad (5)$$

$$\sum_{\forall i \text{ such that } ij \in E, i \neq s} X_{ij} = 1 \forall j, j \neq s, t \quad (6)$$

$$X_{ij} \text{ should be positive integer,} \quad (7)$$

where

- R = Number of Resources
- C_{ij} = Switching Cost of an Edge ij
- ($Cost = 0$ if either $i \in s, t$ or $j \in s, t$)
- ($Cost = -W$ if (i, j) is a pair of duplicated nodes)

The above formulation can be implemented as a min-cost flow problem and then solved optimally under the assumption of single architecture.

As illustrated in Figure 1(D), the algorithm results in exactly R discrete paths from sink to source. These paths are verified by bold lines in Figure 1(D). Each path corresponds to a unit of flow. Each unit of flow represents a resource and the edges that this flow traverses decide the switching activity of this resource. Note that duplicated nodes with a connecting edge of capacity 1 ensure discrete paths in the final solution. Also having an edge weight of $-W$ between each pair of duplicated nodes ensures coverage of all the nodes, as W will be more than the weight of any possible path.

THEOREM 1. *The presented min-cost flow formulation ensures coverage of all the nodes in the end.*

PROOF. Assume R is the given number of resources. R is always chosen to be greater than or equal to the number of operations of the same type in any clock cycle to be able to generate a feasible solution.

Using the min-cost flow formulation we obtain R paths. Now assume there is an uncovered node in a certain clock cycle that cannot be augmented in any of the paths. At least one of the paths should be able to include the uncovered node for that clock cycle. Because R the number of paths is definitely greater than or equal to the number of operations in any clock cycles. The cost of the new path that includes the node is $-W$ units smaller than the cost of the previous path since the two paths differ only in the uncovered node. This new path together with the rest of the paths creates a new solution of R paths. The cost of this solution will then be smaller than the cost of the min-cost flow solution. This contradicts the assumption that min-cost flow generates the best solution. Therefore the assumption of an uncovered node is wrong and the presented min-cost flow algorithm covers all the nodes. \square

Note that the proposed min-cost flow formulation does not consider reiterative effects. By reiterative effects, we mean multiple runs of the DFG where the outputs of the DFG are feeding back the inputs at each iteration. Assume a min-cost flow solution where we have R paths. In a reiterative case, we assume multiple runs of the program where the output of each path will be feeding back its input in multiple iterations. In a compatibility graph, an operation is compatible with any other operation of the same type that belongs to a clock step larger than its clock step. Once binding is done, operations on a resource will be performed in increasing order of their clock cycle. These are all with the assumption that no back edges exist in the DFG. However, with reiterative assumption, back edges will exist from the output of each path to its input forming a cycle (intra-iteration switching activity). This would violate our assumption of

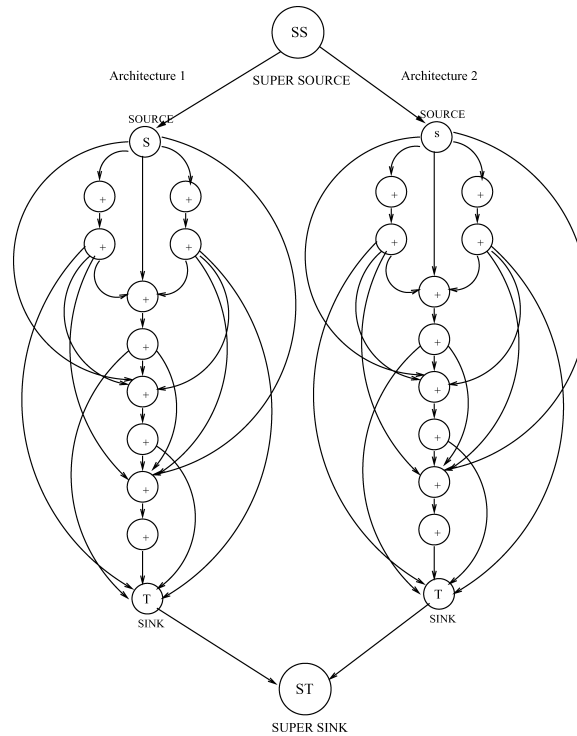


Fig. 2. Multiple-architecture low-power binding.

compatible operations. Kruse et al. [2001] considers low-power binding with reiterative effects and discusses how to obtain optimal solutions in such case.

Also this min-cost flow formulation focuses on a single architecture. By architecture, we mean different ways of implementing any operation on hardware. As an example, an adder can be implemented as carry look ahead or ripple carry. An extension of the single-architecture case finds the best architecture from a library of architectures available for a resource type such that the overall switched capacitance is minimized.

In order to solve this problem, the formulation shown in Figure 1(C) needs to be extended to consider multiple architectures as described in Srivastava [2002] and Raghunathan and Jha [1995].

As Figure 2 shows, let us suppose there are \mathbf{M} different architectures and as before we also have a resource constraint (total number of resources available) of \mathbf{R} . The problem is binding of operations together on \mathbf{R} resources and deciding the architecture for each resource such that the overall switched capacitance is minimized. To address the issue of different architectures, the graph shown in Figure 1(D) is replicated \mathbf{M} times (Figure 2). Hence a node i in the original compatibility graph occurs \mathbf{M} times in this graph. Each s node in the graph is connected to a node called Super-source and each t node is connected to a Super-sink. We need to send \mathbf{R} units of flow from the Super-source to Super-sink such that the overall cost is minimized. The constraints force us to use an

operation in exactly one resource. Each of the replicated subgraphs represent a particular architecture.

The cost values of an edge in that subgraph represent the estimated switched capacitance of having the source and sink of the edge as predecessors and successors on that particular architecture. The cost of the same edge in some other subgraph will be decided by the corresponding architecture. The objective and constraints are formally described below. In order to better understand the formulation, we introduce the following notation. If n is a node in the original compatibility graph (refer to Figure 1(B)), then it gets replicated M times. We denote each of the replicates as n^1, n^2, \dots, n^M .

$$\text{Minimize } \sum_{ij \in E} X_{ij} C_{ij} \quad (8)$$

$$\sum_{\forall j \text{ Super-source } j \in E} X_{\text{Super-source } j} = R \quad (9)$$

$$\sum_{\forall j \text{ Super-sink } j \in E} X_{j \text{ Super-sink}} = R \quad (10)$$

$$0 \leq X_{\text{Super-source } j} \leq R \quad \forall j \text{ Super-source } j \in E \quad (11)$$

$$0 \leq X_{j \text{ Super-sink}} \leq R \quad \forall j \text{ Super-sink } j \in E \quad (12)$$

$$\sum_{k:1..M} \sum_{\forall in^k \in E} X_{in^k} = 1 \quad \forall n \quad (13)$$

$$\sum_{k:1..M} \sum_{\forall n^k j \in E} X_{n^k j} = 1 \quad \forall n \quad (14)$$

$$\sum_{\forall in^k \in E} X_{in^k} = \sum_{\forall n^k i \in E} X_{n^k i} \quad (k : 1..M, \forall n) \quad (15)$$

$$X_{ij} \text{ should be positive integer} \quad (16)$$

Equations (9), (10), (11), and (12) represent the resource constraints. Equations (13), (14), and (15) force each operation in the scheduled DFG to be in exactly one resource. Again this formulation does not consider reiterative effects. The problem is NP-complete as a general case of the problem discussed in Kruse et al. [2001]. It could be solved optimally using any ILP solver.

Even though some previous papers have considered the extended version of the problem, no work has been done to provide fast solutions with negligible error for the generalized low-power binding problem. In Sections 4, a very high-speed and accurate algorithm motivated by the min-cost flow formulation in Chang and Pedram [1995] is proposed to solve the generalized resource binding problem. Section 5 presents another graph-based alternative. Our results are compared to the ILP solution resulted from the formulation above for multiple architectures.

4. IRBINC: ITERATIVE RESOURCE BINDING ITERATIVE NODE COVERAGE

Given \mathbf{M} different architectures and \mathbf{R} different resources of the same type, consider running the min-cost flow algorithm iteratively for \mathbf{M} times for different architectures. The edge costs in each graph relates to its corresponding architecture. From each graph, \mathbf{R} disjoint paths will be obtained.

The goal here is to select up to \mathbf{R} disjoint paths that would cover all the nodes with the least cost from the available $\mathbf{R} \times \mathbf{M}$ set. The following problem can be transformed to a maximal weighted independent set as follows: Assume a new graph is constructed where each path is represented as a node and an edge is inserted between any two nodes in this graph if their corresponding paths overlaps. Each node has a weight that equals $-1 \times$ the corresponding path cost. Solving maximal weighted independent set on this graph corresponds to choosing the set of minimum-cost nonoverlapping paths. Weighted maximal independent set is an NP-Complete problem. Many heuristics have been proposed to efficiently solve this problem [Halldórsson 1999; Boppana and Halldórsson 1990; Niedermeier and Rossmanith 2000] that any of them can be used here.

Here, we explain our Iterative Resource Binding, Iterative Node Coverage (IRBINC) algorithm. Once \mathbf{R} disjoint paths are generated for each architecture, IRBINC iteratively selects the path with the smallest weight from the set of available paths. Whenever a path is selected and added to the solution set, all of the overlapping paths from the available set will be removed. IRBINC iterates until the available set becomes empty meaning that each path that is not selected has at least one overlap with a path in the final solution set.

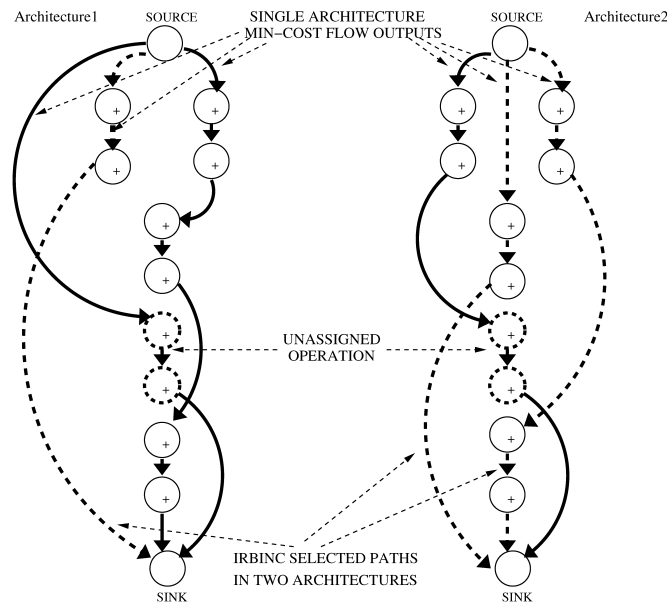
Consider Figure 3(A). The single architecture flow formulation of the previous section results in 3×2 paths here for two architectures and three resources (six bold and dashed paths). After the iterative path selection, three final paths are obtained illustrated by the dashed lines. As can be seen, the first path is from the first architecture and the second and third ones are from the second architecture for this example.

At this point, up to \mathbf{R} paths are obtained, but there is still possibility of some uncovered nodes, which are the operations that have not yet been assigned to any resource, as the example of Figure 3(A) shows.

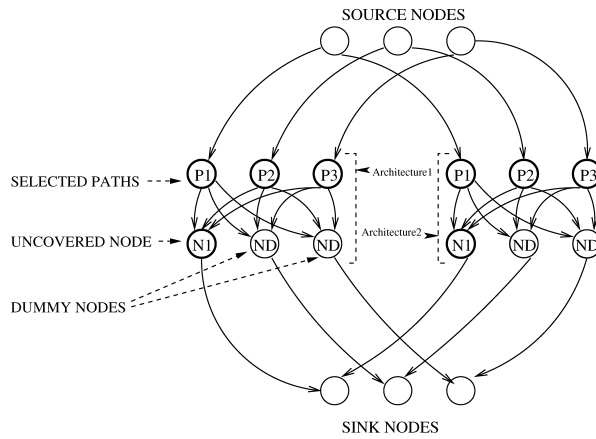
We proposed a node coverage algorithm that simultaneously assigns the uncovered nodes of the same clock cycle in the DFG to the resources obtained before. It runs once for any clock step that contains uncovered operation(s). This is iteratively done until all the nodes are covered. At each run, a group of uncovered operations that belong to the same clock cycle are considered.

The strategy of the node coverage algorithm is to assign the uncovered nodes into these \mathbf{R} paths with the smallest overall cost following another flow formulation. The node coverage algorithm described below runs iteratively until all the uncovered operations are assigned.

Node Coverage Algorithm. Given up to \mathbf{R} selected paths (assigned resources) and \mathbf{P} uncovered nodes (unassigned operations) of the same clock cycle in the DFG, a new graph is constructed as follows. Consider a node for each path and a node for each uncovered operation. If the number of uncovered nodes is less than the number of resources ($\mathbf{P} < \mathbf{R}$), dummy nodes will be added to make it equal.



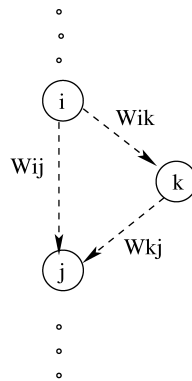
(A) ITERATIVE SINGLE ARCHITECTURE MIN-COST FLOW SOLUTION



(B) COVERING UNASSIGNED OPERATIONS

Fig. 3. Node coverage algorithm.

An edge goes from a “path-vertex” to an “operation-vertex”, if the node can be augmented in the corresponding path [there is no operation scheduled on the path in the related clock cycle]. As Figure 3(B) illustrates, the described structure is repeated M times. R source and R sink nodes are added. Each source node corresponds to a path and each sink node corresponds to an uncovered operation. Directed edges of 0 cost goes from each source node to its corresponding path-vertex and from each of the $R \times M$ operation-vertices to the corresponding sink node. All the edge capacities are unity to ensure disjoint solutions.



$$W_{\text{PATH_NEW}} = W_{\text{PATH_OLD}} - W_{ij} + W_{ik} + W_{kj}$$

Fig. 4. Updating a path cost.

The edge costs are modified correspondingly for each architecture. The cost of an edge from a path-vertex to an operation-vertex is the cost of the new path obtained after augmenting the node as Figure 4 illustrates. The cost of an edge to a dummy node is simply the path cost.

By sending one unit of flow from each source node to any of the sink nodes such that all the costs are minimized, up to R new paths are obtained that cover the P uncovered operations simultaneously.

The edge from a path-vertex to an operation-vertex having a positive flow decides the coverage of the corresponding operation in the corresponding path. The subgraph at which this edge occurs decides the architecture. Note that if the actual number of paths from the iterative resource binding is smaller than the number of resources, some of the path-vertices will be “dummy paths”. A dummy path covering an operation-vertex corresponds to a new resource that will be assigned to an unassigned operation. The node coverage algorithm considers the set of unassigned operations simultaneously for different architectures and resources. It runs iteratively for each set of unassigned operations of the same clock cycle in the DFG until all the nodes are covered.

THEOREM 2. *The node-coverage algorithm covers the uncovered nodes in maximum R paths; that is, It does not increase the total number of paths more than R .*

PROOF. Two things are noted in this proof.

- (1) R is always given to be at least the minimum number of resources to create a feasible solution. This means R for any resource type is always greater than or equal to the number of operations (of that type) at any clock step.
- (2) The rest is proof by contradiction. Assume in the final selected paths for a certain clock cycle there is an uncovered node that cannot be augmented in any of the paths. If the number of paths initially generated by IRBINC is less than R , the node can be covered in a new path (dummy node for a new path in the node coverage algorithm). However, if the number of paths

is R , the node should be able to get included in at least one of the paths. If not, this would contradict the assumption that R is greater than or equal to the number of operations (of same type) at any clock step. Since already R operations are assigned to R paths for that clock cycle, there cannot be anymore operations left out in that clock cycle. Therefore, at least one path exists that can include the node in that clock cycle. The node coverage algorithm determines the min-cost path among all such. Therefore, any such uncovered node will be covered in at most R paths. \square

Once a valid solution is generated by covering all the uncovered nodes, additional optimization is done to improve upon the generated solution. We refer to this as IRBINC-REF. Since IRBINC inherently lacks a global view in covering the uncovered nodes from one cycle to another, once the uncovered nodes are covered using the node-coverage algorithm, we continue the optimization to improve upon the generated solution. In the refined version, a set of moves are iteratively made by exchanging the bindings of the current solution at different clock cycles. This is done by traversing the graph from the first to the last clock cycle. At each clock step, potential exchanges of the bindings of operations are explored. If an exchange of binding results in an improvement in the overall cost, the move is accepted. However, if the cost is not improved, the move can still be accepted in order to avoid the optimization getting blocked in a local minima. This is done iteratively (for k rounds defined by user) and, in the end, the minimum cost solution encountered up to that point is chosen as the final solution. One way to consider potential exchanges of operations in a clock cycle is by using the presented node coverage algorithm and assuming all the operations in that clock cycle to be uncovered. This refined version of the algorithm results in some additional improvement.

Complexity Analysis. The single architecture min-cost flow is of $O(n^3)$ for n operations and runs iteratively M times for each architecture. Iterative path selection is $O(RM \log(RM))$ for $R \times M$ paths. Each iteration of the node coverage algorithm is of $O((2R + 2RM)^3) = O((RM)^3)$, as Figure 3(B) illustrates, the total number of nodes for any such graph is $2R$ for the source and sink nodes and $2R$ nodes for each architecture. The node coverage algorithm iterates at most T clock steps. Therefore the running time of the IRBINC algorithm is described as $O(Mn^3 + RM \log(RM) + TR^3 M^3) = O(Mn^3 + TR^3 M^3)$, which is a polynomial expression.

IRBINC considers selection of different architectures of each resource at the time of resource binding. Our experimental results show a very high quality solution and great speed up compared to the optimal ILP discussed in Section 3. In Section 5, another heuristic with similar characteristics is presented.

5. CTDA: CONSTRUCTIVE TOP-DOWN ASSIGNMENT

We propose another approach in which the solution is generated in a constructive manner. The scheduled DFG is traversed in a top-down fashion from clock step 1 to clock step T , hence the name *Constructive Top-Down Assignment*. Starting from clock-1, we have a set of operations in clock-1 that need to be bound onto R resources. We arbitrarily make this assignment to the available

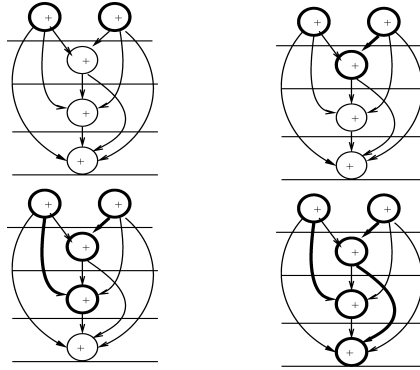


Fig. 5. Development of paths in CTDA.

resources. The clock steps are traversed iteratively in a top-down fashion. Let us suppose we have traversed the first i clock steps. At this stage, we have an intermediate solution in which a set of operations belonging to the first i clock steps have already been bound to \mathbf{R} resources. Clock $i + 1$ contains a set of operations that need to be bound onto these \mathbf{R} resources. This can be done using the *node coverage algorithm* presented in the previous section. That formulation assigned operations in the same clock step to a set of \mathbf{R} resources on which some operations have already been bound. There is one key difference though. The cost of an edge connecting a path-vertex to an operation-vertex will be decided by the cost of the compatibility edge between the operation and the last operation bound (in terms of clock step) onto that resource. This formulation simultaneously assigns operations in the same clock step to available resources and also selects the architecture with power as the optimization objective. Then, we proceed to the next clock step, $i + 2$, and the same thing is repeated.

Figure 5 illustrates running of the algorithm. The operations in bold are the ones that have been assigned, the bold edges signify the paths or sequence of operations on the corresponding resource. It can be seen that operations are considered one clock at a time in a top-down fashion. Please note that this strategy is similar to the left-edge algorithm [Kruse et al. 2001] on interval graphs. The scheduled operations could be considered as intervals in a temporal sense with finite lifetimes (starting time to ending time) indicating lengths or intervals of nodes. We proceed on this so-called interval graph in a left edge (or top down) fashion, solving a local binding problem optimally at each step. The global algorithm is of course a heuristic.

Complexity Analysis. At most, R operations are covered at each clock cycle. Assuming T clock steps, the node coverage algorithm will be called at most T times. Each iteration of the node coverage algorithm is of $O(R^3M^3)$ as described in the previous section. Therefore, the time complexity of CTDA is $O(T \times R^3M^3)$.

6. CONSIDERING PIPELINED DESIGNS/MULTIPLE ITERATIONS

When considering pipelined designs, the bindings of resources can differ (or can get exchanged as in Chang and Pedram [1996]) from one iteration to

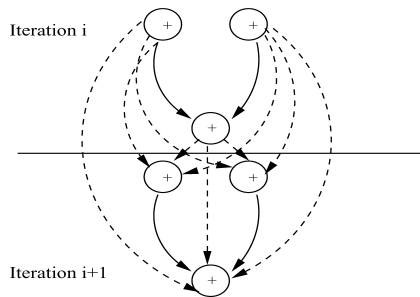


Fig. 6. Construction of compatibility graph for more than one iteration.

another iteration of the DFG. The proposed techniques only consider the resource binding problem for one iteration.

In order to consider pipelined designs, the compatibility graph in ILP, IRBINC or CTDA is expanded to take the switching activity of interiteration edges into account. Considering k iterations of the input DFG, the comparability graph constructed for a fixed architecture is replicated sequentially for k times. The i th replication represents the i th iteration. Within each iteration the compatibility graph is identical to the original. The cost of each edge represents the switching activity of the resource if the two operations connected by the edge are performed consecutively on the resource. To consider interiteration switching activity, additional compatibility edges are added from all operations in iteration (replication) i to all operations in iteration (replication) $i + 1$. The costs of these edges similarly reflect the switching activity of a resource if the two operations belonging to two consecutive iterations and connected through the edge are performed consecutively on the resource. Figure 6 shows an example for a simple graph with two iterations.

Once the new compatibility graph is constructed for k iterations the same philosophy can be applied for the heuristics or the general ILP formulation. In the generic ILP formulation, each architecture consists of a compatibility graph for k iterations. In IRBINC, once the compatibility graph with k iterations is constructed for each architecture where the edge costs are for the corresponding architecture.

In CTDA, a generic compatibility graph is constructed for k iterations and as the nodes are covered using the node coverage algorithm at each clock cycle, different costs due to different architectures are considered. The algorithms in all the cases would stay the same.

This approach effectively considers the possibility of processing multiple iterations simultaneously. Other approaches that share similar philosophy, such as Chang and Pedram [1996], can be used.

7. RESULTS

The experimental environment was setup similar to Srivastava [2002]. The design flow is shown in Figure 7. We take functions written in C and extract data flow graphs from them using SUIF/Machine-SUIF. These DFGs are then scheduled using a path based scheduler [Ogrenci-Memik et al. 2001]. We also

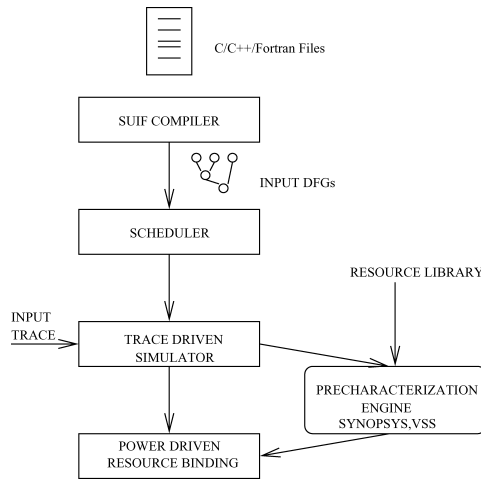


Fig. 7. Experimental flow.

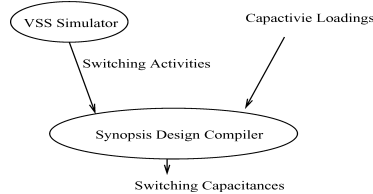


Fig. 8. Calculation of switching activities.

take an input trace which is a representative of the input statistics for the DFG. Based on this trace, the DFG is simulated and the values for all the internal variables of the DFG are computed. This information along with the scheduling information is given to Synopsys Design Compiler and the VSS simulator to create switched capacitance values. Figure 8 shows how the edge weights in the compatibility graphs are calculated. Basically the VSS simulator generates the switching activities for any pair of operations of the same type. These together with capacitive loadings will be fed to Synopsys Design Compiler. Using these values the switched capacitances are calculated. These values correspond to the edge costs for the compatibility graph of the corresponding schedule.

We used the Media bench suite [Lee et al. 1997] together with some HLS benchmarks to extract the data flow graphs. Our experiments are done on a Solaris 2.7 operating system (Sparc edition) with 256 M memory. Here, power optimization for only additions are considered from different operation types in the DFG.

Table I illustrates the performance of the two proposed methods compared to the ILP case for multiple architectures formulated in Section 3. For each benchmark the number of operations (additions) are reported in column 2. The number of available resources is reported in column 3. Both the IRBINC and the CTDA result in great speed-ups up to 13.7. Even though the speed-up of the two methods are similar, in the case of motion-3, noisest-2 and jdmerge-3, IRBINC

Table I. Performance of the Proposed Techniques

Bench	#Ops	#Res	IRBINC		CTDA		ILP
			Time(Sec)	Speedup	Time(Sec)	Speedup	Time(Sec)
fft-2	16	3	1.00	3.87	0.73	5.30	3.87
motion-2	21	3	0.76	10.70	1.20	6.75	8.13
motion-3	21	3	0.80	10.12	1.73	4.68	8.10
noisest-2	16	3	0.73	5.89	4.48	0.96	4.30
ecb-enc-4	22	3	1.17	13.7	1.30	12.33	16.03
dct	13	3	1.03	1.52	0.70	2.24	1.57
ellipt	26	3	1.77	2.56	1.57	2.88	4.53
jdmerge-1	23	3	0.73	7.85	1.00	5.73	5.73
jdmerge-3	29	3	0.87	11.49	2.03	4.93	10.00
jdmerge-4	18	3	0.63	9.36	0.80	7.37	5.90
fir	10	2	0.80	1.25	0.40	2.50	1.00
jctrans2	16	3	0.70	5.71	0.60	2.40	4.00
Average					7.0		4.8

Table II. Penalty of the Proposed Techniques

Bench	IRBINC		CTDA		ILP
	Pnlty	SwitchCap	Pnlty	SwitchCap	SwitchCap
fft-2	2.49	130.78	0.93	128.79	127.60
motion-2	0.01	163.33	2.19	166.91	163.32
motion-3	0	158.11	2.48	162.04	158.11
noisest-2	0.47	107.98	1.09	108.64	107.47
ecb-enc-4	1.45	196.76	0.57	195.04	193.94
dct	0.2	79.78	0.64	80.13	79.62
ellipt	1.19	179.54	2.28	181.48	177.43
jdmerge-1	0	161.07	0	161.07	161.07
jdmerge-3	0.46	219.00	0.70	219.52	217.99
jdmerge-4	2.50	138.53	2.58	138.64	138.15
fir	0.67	59.75	0	59.35	59.35
jctrans2	0	66.46	0	66.46	66.46
Average	.79		1.12		

results in greater speed-ups. The proposed methods have better performance on large benchmarks. As can be seen in the case of DCT, which represents a small benchmark, the speed-up is really small. The average speed-up from the 12 benchmarks is 7.00 and 4.84 for IRBINC and CTDA respectively. Table II shows the power cost penalty of both of the schemes to be really small. The maximum penalty is only 2.58% in the case of jdmerge-4. Average penalties are 0.78% and 1.12% for IRBINC and CTDA respectively.

We also performed experiments with larger benchmarks by unrolling the FIR and FFT benchmarks with different factors. The unrolling was done by replicating the original benchmark sequentially, so that for the k -unrolled version, the number of clock cycles was k times of the original. The number of operations (additions) of the k -unrolled version is k times more than the original benchmark. Figures 9 and 10 compare the switched capacitance and run-time in ILP, IRBINC and IRBINC-REF as a function of the unrolling factor (or growth size of the benchmark) for the FIR and FFT benchmarks respectively. These figures

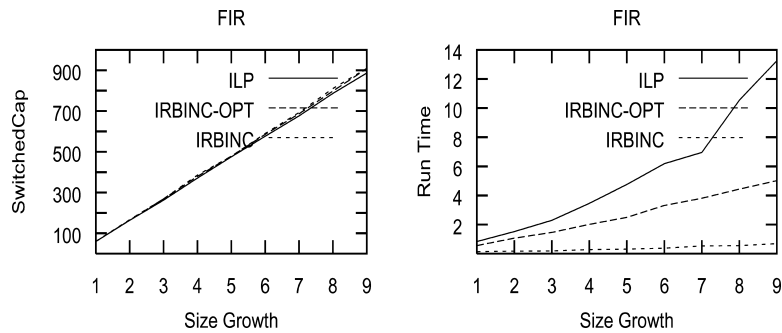


Fig. 9. Comparison of switched capacitance & run-time with different unrolled factors.

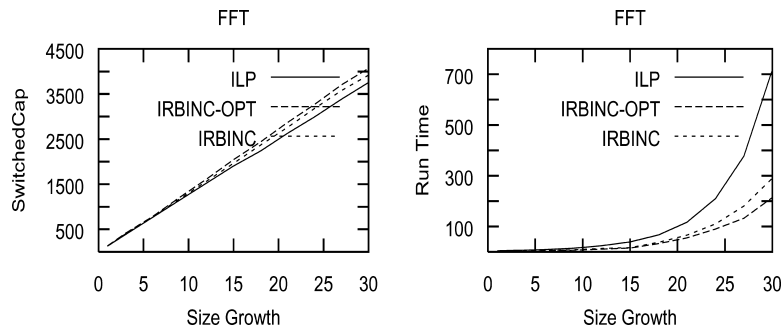


Fig. 10. Comparison of switched capacitance & run-time with different unrolled factors.

show that as the size of the benchmark is increased, the run-time of the ILP approach increases with a much larger rate while the growth in penalty of switched capacitance in IRBINC and IRBINC-REF is much smaller. The run-time improves consistently as the benchmark size grows for both IRBINC and IRBINC-REF when compared to ILP.

8. CONCLUSION

This article formally discusses the low-power binding problem for multiple resources and architectures. It introduces two fast polynomial-time alternatives to the optimal ILP approach. They result in a maximum speed-up of 13.7 with a maximum deviation of only 2.5% from the optimal solution. The proposed algorithms provide much better speed-ups for larger benchmarks which should make them very suitable in real-world applications. In the end, we would like to point out that the above formulation can be useful in similar optimization problems that more than one choice/implementation exist for the inputs.

REFERENCES

- BOPANA, R. AND HALLDÓRSSON, M. M. 1990. Approximating maximum independent sets by excluding subgraphs. In *Proceedings of the SWAT'90 2nd Scandinavian Workshop on Algorithm Theory* 447, pp. 13–25.
- BRINGMANN, O. AND ROSENSTIEL, W. 1997. Resource sharing in hierarchical synthesis. In *Proceedings of the International Conference on Computer Aided Design* (Nov.). pp. 318–325.

- CHANDRAKASAN, A. P., SHENG, S., AND BRODERSEN, R. W. 1992. Low power CMOS digital design. *IEEE J. Solid State Circ.* (Apr.) pp. 472–484.
- CHANG, J.-M. AND PEDRAM, M. 1995. Low power register allocation and binding. In *Proceedings of the Design Automation Conference* (June). pp. 29–35.
- CHANG, J.-M. AND PEDRAM, M. 1996. Module assignment for low-power. In *Proceedings of the European Design Automation Conference* (Sept.). pp. 376–381.
- FANG, Y. M. AND WONG, D. F. 1994. Simultaneous functional-unit binding and floorplanning. In *Proceedings of the International Conference on Computer-Aided Design* (Nov.). pp. 317–321.
- HAFER, L. AND PARKER, A. 1982. Automated synthesis of digital hardware. *IEEE Trans. Comput.*, Feb.
- HALLDÓRSSON, M. M. 1999. Approximations of weighted independent set and hereditary subset problems. In *Proceedings of the 5th Annual International Conference on Computing and Combinatorics, Lecture Notes in Computer Science*. Springer-Verlag, New York, pp. 261–270.
- HASHIMOTO, A. AND STEVENS, J. 1971. Wire routing by optimizing channel assignment with large apertures. In *Proceedings of the 8th Design Automation Workshop*. pp. 155–169.
- HERRMANN, D. AND ERNSTL, R. 1999. Improved interconnect sharing by identity operation insertion. In *International Conference on Computer-Aided Design* (Nov.). pp. 489–492.
- KRUSE, L., SCHMIDT, E., JOCHENS, G., STAMMERMANN, A., SCHULZ, A., MACII, E., AND NEBEL, W. 2001. Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs. *IEEE Trans. VLSI Syst.* (Feb.). pp. 3–14.
- LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the International Symposium on Microarchitecture*.
- NIEDERMEIER, R. AND ROSSMANITH, P. 2000. On efficient fixed parameter algorithms for WEIGHTED VERTEX COVER. In *Proceedings of the International Symposium on Algorithms and Computation*, pp. 180–191.
- OGRENCI-MEMIK, S., BOZORGZADEH, E., KASTNER, R., AND SARRAFZADEH, M. 2001. A super scheduler for embedded reconfigurable systems. In *Proceedings of the International Conference on Computer Aided Design* (Nov.). pp. 391–394.
- RAGHUNATHAN, A. AND JHA, N. 1995. An ILP formulation for low power based on minimizing switched capacitance during datapath allocation. In *Proceedings of the IEEE Symposium on Circuits and Systems*.
- RAJE, S. AND BERGAMASCHI, R. A. 1977. Generalized resource sharing. In *Proceedings of the International Conference on Computer Aided Design* (Nov.). pp. 326–332.
- SRIVASTAVA, A. 2002. Predictability: Definition, analysis and optimization. In *Proceedings of the International Conference on Computer Aided Design*, Nov.
- TSENG, C. J. AND SIEWIOREK, D. 1986. Automated synthesis of data paths in digital systems. *IEEE Trans. Comput. Aid. Des.* (July).

Received August 2003; revised June 2004, December 2004, and May 2004; accepted July 2005