Performance-Driven Placement for Dynamically Reconfigurable FPGAs

GUANG-MING WU
Nan-Hua University,
JAI-MING LIN
Realtek Semiconductor Corp.,
and
YAO-WEN CHANG
National Taiwan University

In this article, we introduce a new placement problem motivated by the Dynamically Reconfigurable FPGA (DRFPGA) architectures. Unlike traditional placement, the problem for DRFPGAs must consider the precedence constraints among logic components. For the placement, we develop an effective metric that can consider wirelength, register requirement, and power consumption simultaneously. With the considerations of the new metric and the precedence constraints, we then present a three-stage scheme of partitioning, initial placement generation, and placement refinement to solve the new placement problem. Experimental results show that our placement scheme with the new metric achieves respective improvements of 17.2, 27.0, and 35.9% in wirelength, the number of registers, and power consumption requirements, compared with the list scheduling method.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*; B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Algorithms, Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Computer-aided design of VLSI, dynamically reconfigurable, layout, placement, field-programmable gate array

1. INTRODUCTION

Improving logic efficiency by time-sharing, Dynamically Reconfigurable FPGAs (DRFPGAs) have gained much attention recently. In a DRFPGA, a virtual large

The work of G.-M. Wu was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-91-2215-E-343-001.
Authors' addresses: G.-M. Wu, Dept. of Information Management, Nan-Hua University, Dalin Chiayi 22, Taiwan; email: gmwu@mail.nhu.edu.tw; J.-M. Lin, Realtek Semiconductor Corp., No. 2, Industry E. Rd. IX, Science-Based Industrial Park, Hsinchu, Taiwan; email: jarvis@realtek.com.tw; Y.-W. Chang, Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan; email: ywchang@cc.ee.ntu.edu.tw.

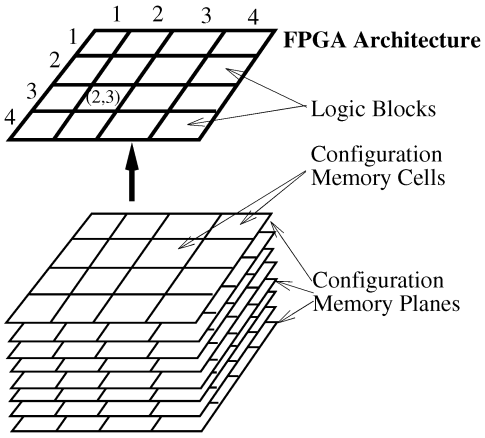

Fig. 1. Xilinx dynamically deconfigurable FPGA configuration model.

design is partitioned into multiple *stages* (or partitions) to share the same smaller physical device as that occupied by a traditional FPGA. Various architectures have been proposed, such as the Xilinx model [Trimberger 1997], the Dynamically Programmable Gate Array [Brown et al. 1995], and the Virtual Element Gate Array [Jones and Lewis 1995]. In these models, on-chip SRAM bits are programmed to record the configuration of each stage. Dynamic reconfiguration of logic blocks and wire segments can be performed by reading the on-chip SRAM bits of each configuration in order.

Figure 1 shows the Xilinx DRFPGA configuration model [Trimberger 1997]. The Xilinx DRFPGA emulates a single large design in multiple configurations. Each configuration can be stored in a *configuration memory plane* (CMP) which consists of a two-dimensional array of *configuration memory cells* (CMCs). In each *microcycle*, the SRAM bits of the corresponding configuration are loaded into the DRFPGA, and the configurable logic blocks (CLBs) are reused to evaluate combinational logic. One pass through all microcycles is called a *user cycle*. The target architecture consists of an array of augmented XC4000E-style CLBs [Trimberger 1997]. Each CLB includes a set of *microregisters* (*MRs*) to hold the CLB results between configurations. Every CMC of the original FPGA is packed by eight inactive memory cells. MRs not only store the intermediate values of combinational logic for use in later microcycles, but also hold latch values for use in the next user cycle. A microcycle starts with saving all the CLB results of the previous microcycle in MRs, and then a new configuration is loaded into the active configuration memory. The loading process is called *flash reconfiguration*.

Because the logic and interconnect needed for a circuit are time-multiplexed on a DRFPGA, the partitioning and placement problems are different from traditional ones. The major difference is that the execution order of circuit elements must follow the precedence constraints in the DRFPGAs. The DRFPGA partitioning problem has been studied in the recent literature [Chang and Marek-Sadowska 1997, 1998; Chao et al. 1999; Liu and Wong 1998; Wu et al. 2001]. In these partitioning algorithms, the major objective is minimizing the cut size

(MRs) between microcycles or user cycles; nevertheless, the exact number of MRs needed will be known at the placement stage. (It is possible to refine the partitioning results in the placement stage.) Moreover, there are some objectives that must be considered in placement, such as wirelength, which cannot be considered in partitioning. Therefore the placement for DRFPGAs is an important and valuable problem.

In traditional FPGAs, if we want to implement a circuit, the circuit must be loaded into a FPGA at the same time. Therefore the major concern in the placement for traditional FPGAs is the total wirelength that will have a great effect on routing. Due to the reuse of logic and interconnect, the placement problem for DRFPGAs is quite different from the traditional one. Unlike traditional FPGAs, the order of the execution of nodes must satisfy the precedence constraints in a DRFPGA. As in the statement described in previous paragraphs, we need some MRs to store the values between microcycles (or user cycles) during a circuit executed in a DRFPGA. Consequently, the number of MRs used in the placement is an important consideration for DRFPGAs. We refer to the *lifetime* of a node in a DRFPGA as the duration from the stage where the node is assigned to the stage when it is last used. The intermediate value of a node must be stored in an MR during its lifetime. The values of several nodes can be stored in the same MR if the lifetimes of the nodes do not overlap. In contrast, if there are two combinational or latch nodes placed in the same position on different memory planes and their lifetimes overlap, then their results cannot be stored in the same memory space of an MR. Also, the number of nodes whose lifetimes overlap in the same position cannot exceed the MR capacity—the *MR-capacity constraint*. The traditional FPGA placement problem has been studied to some degree in the literature [Alexander et al. 1995; Chang et al. 1994; Chen et al. 1995; Lee and Wu 1995; McMurchie and Ebeling 1995; Togawa et al. 1994]. The previous work applies some metrics to estimate wiring cost (wirelength, delay, etc.) of a net; the metrics are usually incorporated in popular wiring estimation schemes such as semiperimeter, Steiner tree, minimum spanning tree, and the like. Due to the precedence and MR-capacity constraints, however, those metrics cannot apply to DRFPGA placement.

In this article we introduce a new placement problem motivated by the DRFPGA architectures. For the DRFPGA placement, we develop a new metric that can simultaneously consider wirelength, MR usage, and power consumption under the precedence constraints. (The previous works only deal with wiring cost but not MR usage, power consumption, and precedence constraints.) With the considerations of the new metric and the precedence constraints, we then present a three-stage scheme of partitioning, initial placement generation, and placement refinement to solve the new placement problem for DRFPGAs. The first stage partitions a circuit into k subcircuits without violating the precedence constraint, where k is the number of CMPs in a DRFPGA. The k-way DRFPGA partitioning method is an extension of the FM [Fidducia and Mattheyses 1982] balanced bipartitioning. In the partitioning, we reduce the length of lifetime for each node as much as possible, as the length of lifetime is closely related to the number of MRs required. The second stage employs a constructive method to obtain an initial placement; nodes are placed in the

decreasing order of the percentages of their already placed neighbors. The last stage applies a simulated annealing approach to improve the initial placement. Experiments with the benchmark circuits used in Chang and Marek-Sadowska [1998] show that our placement scheme with the new metric achieves respective improvements of 17.2, 27.0, and 35.9% in wirelength, the number of registers, and power consumption requirements, compared to the list scheduling method.

The remainder of this article is organized as follows. Section 2 formulates the new placement problem. Section 3 presents the new metric for the DRFPGA placement. Section 4 proposes the three-stage placement algorithm. Section 5 shows the experimental results, and conclusions are given in Section 6.

2. PROBLEM FORMULATION

In this article, all circuits are preprocessed by a lookup table-based (LUTs) technology mapper [Sentovich and Singh 1992] and thus the circuit components are composed of lookup tables, latches, and netlists. We represent a circuit by a directed hypergraph $G = (V, E)$, where V is the set of LUTs and latches and E is the set of nets. We denote a net e by $e = (v_1 \rightarrow \langle v_2, v_3, \ldots, v_n \rangle)$, where v_1 is the *fanout* node whose output signal is the input signal to v_j $(2 \leq j \leq n)$, and v_j $(2 \leq j \leq n)$ is the *fanin* node whose input signal is the output signal from v_1. The set E can be divided into two subsets E_c and E_f according to the type of fanout node. A net $e \in E_c$ (E_f) if the fanout node of e is an LUT (latch) node.

For a DRFPGA, a circuit is placed into several CMPs such that the logic in different CMPs temporally shares the same physical CLBs by setting the CMPs active in order. To ensure the correct results of a circuit in a user cycle, the nodes must be evaluated in the proper order. According to the Xilinx architecture, the following *precedence constraints* must be satisfied.

— Each LUT node must be placed in a CMP no later than all its output nodes.
— Each latch node must be placed in a CMP no earlier than all its input nodes. (This ensures that latch input values are calculated before they are stored.)
— Each latch node must be placed in a CMP no earlier than all its output nodes. (This ensures that all of the nodes use the same value of the latch—the value of the latch from the previous user cycle.)

The above constraints define a partial temporal ordering on the nodes in the circuit. Let $Pre(v)$ be the precedence of a node v. For two nodes v and u, we define $Pre(v) \preceq Per(u)$ if v must be placed no later than u. Let $s(v) = i$ if node v is assigned to the CMP i. $s(v) \leq s(u)$ if $Pre(v) \preceq P(u)$. The placement with the precedence constraint is called *precedence-constrained placement* (*PCP*).

Two nodes are said to be *related* if they are placed in the same CLB of two different CMPs. The *lifetime* of a node is the duration from the CMP where it is assigned to the CMP where it is last used. A node in its lifetime is called a *live node*; that is, the data of the node must be stored in an MR for later use. For the fanout node v_1 of a net $e = (v_1 \rightarrow \langle v_2, v_3, \ldots, v_n \rangle)$, and if $e \in E_c$, the lifetime of v_1 is from the CMP $s(v_1)$ to the CMP $\max\{s(v_j) | 2 \leq j \leq n\}$, and if $e \in E_f$, the lifetime of v_1 is from the CMP $s(v_1)$ to the last CMP and from the CMP 1 to the CMP $\max\{s(v_j) | 2 \leq j \leq n\}$, because the output of a latch node is used

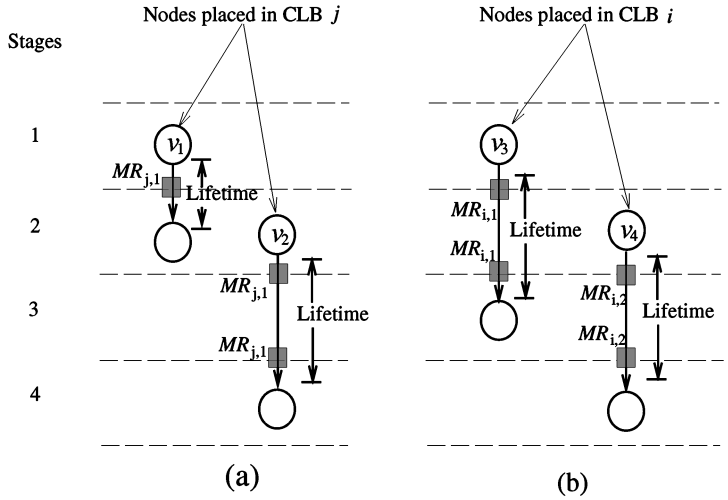

Fig. 2. (a) Because the lifetimes of related nodes v_1 and v_2 do not overlap, their results can be stored in the same MR; (b) the lifetimes of related nodes v_3 and v_4 overlap, so their results must be stored in different MRs.

in the next user cycle. If there does not exist any net whose fanout node is the node v_1, v_1 has no lifetime. It implies that the data of v_1 do not have to be stored for later use. The result of a node must be stored in an MR during its lifetime; several related nodes can share the same MR if their lifetimes do not overlap, as shown in Figure 2(a). In other words, the results of two related nodes must be stored in different MRs if their lifetimes overlap, as shown in Figure 2(b). Therefore, the lifetimes of nodes affect the number of MRs required.

In Xilinx DRFPGAs, during the reconfiguration stage, first the logic and interconnect array are updated simultaneously from a configuration memory plane. Second, it must settle the signals from the microregisters. The power consumption during reconfiguration and settle signals can be very high [Trimberger 1997]. Therefore we try to consider power consumption in the PCP. For two nodes u and v, if $Pre(u) \preceq Pre(v)$ and u and v are placed in the same CLB of different CMPs, node v can get the result of u immediately from the MR in its own CLB after flash reconfiguration, for example, the case of nodes v_1 and v_2 in Figure 3. If $Pre(u) \preceq Pre(v)$ and u and v are placed in different CLBs of different CMPs, the result of the node u must be passed to the node v by an extra connection during flash reconfiguration (e.g., the case of nodes v'_3 and v'_4 in Figure 3), that is to say, this needs an interconnect and a signal during reconfiguration and will increase the power consumption of the system. The nodes v'_3 and v'_4 are called a *power-consumption pair*. Considering the power consumption in the DRFPGA placement, we prefer to place nodes in the same CLB of CMPs if they have data dependency.

We use the following notation throughout this article.

—$c(u, v)$: A power-consumption pair for nodes u and v.
—C: The set of all power-consumption pairs in the placement.

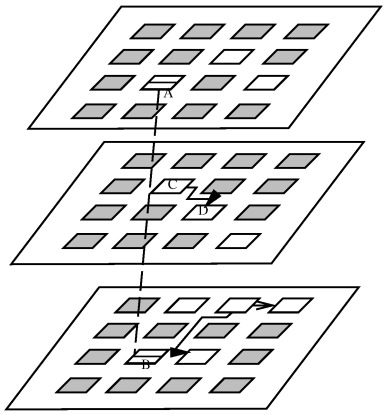

Fig. 3. Placing net ($v_3 \rightarrow \langle v_4 \rangle$) in empty cells, with the other net ($v_1 \rightarrow \langle v_2 \rangle$) being preplaced. Case 1: v_3 and v_4 are placed in the same CMP; only the wirelength must be considered in the metric. Case 2: v_3' and v_4' have wirelength and power consumption penalties, but no wiring and power-consumption penalties. Case 3: v_3'' and v_4'' have wirelength, memory, and power penalties.

— $|C|$: The number of power-consumption pairs in C.
— $B = (P, M)$: P is the set of configuation memory cells (CMCs) and M is the set of MRs.
— $D(B)$: A DRFPGA, where B is the set of $n \times n$ CLBs in the DRFPGA.
— $b_{i,j} \in B$ $(1 \leq i, j \leq n)$: The CLB at the grid location (i, j) in D.
— $p_{k,i,j}$: The CMC at the grid location (i, j) in CMP k.
— $P_k = \{p_{k,i,j} | 1 \leq i, j \leq n\}$: The set of CMCs in CMP k.
— $P = \bigcup_{i \in \{1,2,\ldots,r\}} P_i$, where r is the number of stages (CMPs) in the DRFPGA.
— $m_{i,j} \in M$: The set of MRs needed in $b_{i,j}$.
— $|m_{i,j}|$: The size of $m_{i,j}$ (i.e., the number of MRs needed in $b_{i,j}$).
— $p^t_{k,i,j}$: An LUT cell in $p_{k,i,j}$.
— $p^l_{k,i,j}$: A latch cell in $p_{k,i,j}$. (Each CMC $p_{k,i,j}$ consists of an LUT cell and a latch cell.)

The Precedence-Constrained Placement (PCP) problem is defined as follows.

— *Instance*: A DRFPGA $D(B)$ and a circuit graph $G(V, N)$.
— *Problem*: Assign each LUT node (latch node) to a unique CMC $p^t_{k,i,j}$ ($p^l_{k,i,j}$), where $1 \leq k \leq r$ and $1 \leq i, j \leq n$ so that

1. the total wirelength,
2. $\max\{|m_{i,j}| | 1 \leq i, j \leq n\}$, and
3. $|C|$

are simultaneously minimized, and for any nodes v_1 and v_2, $s(v_1) \leq s(v_2)$ if $Pre(v_1) \preceq Pre(v_2)$.

The first objective considers wirelength. Unlike the traditional placement problem, the estimation of the wirelengths in the PCP must consider two cases. One is that all nodes of a net are assigned to the same CMP. In this case, the

wirelength is estimated by the geometric (Manhattan) distance of the net, the same as the traditional measurement. The other is that the nodes of a net are assigned to different CMPs. For this case, we must project all nodes to the same CMP and then estimate the wirelength as in the previous case. The second objective tries to minimize $\max\{|m_{i,j}||1 \leq i, j \leq n\}$, facilitating the design to fit into a CLB with fewer MRs. Note that the MRs in the CLBs of a DRFPGA are all identical. The third objective is intended to minimize power consumption.

3. THE EFFECTIVE METRIC FOR THE PCP

In this article, we are primarily concerned with the problem of finding an effective metric to guide the low-power precedence-constrained placement. By effective, we mean one that can simultaneously minimize wirelength, MR count, and power consumption for the problem being considered.

In PCP, to achieve good performance, the metric must consider these issues: (1) wirelength, (2) microregister requirement, and (3) power consumption. The metric presented in this article is defined as follows. Let $x(e)$ be the placement of a net e that satisfies the precedence constraints. The cost for $x(e)$, $\Phi(x(e))$, is given by

$$\Phi(x(e)) = \alpha w(x(e)) + \beta h(x(e)) + \gamma o(x(e)), \tag{1}$$

where $w(x(e)), h(x(e)),$ and $o(x(e))$ represent the respective cost functions for wirelength, MR count, and power consumption, and $\alpha, \beta,$ and γ are user-specified parameters. Here, $\alpha + \beta + \gamma = 1$ and $\alpha, \beta, \gamma \geq 0$. Example 1 illustrates several cases of a placement of a two-terminal net and estimates their cost by our metric.

Example 1. In the example shown in Figure 3, we assume that the net $(v_1 \rightarrow \langle v_2 \rangle)$ has been preplaced and $|m_{3,2}|$ is the largest among $|m_{i,j}|$ $(1 \leq i, j \leq n)$ presently. Assuming $\alpha = \beta = \gamma = \frac{1}{3}$, we consider the three cases of the placement for net $e = (v_3 \rightarrow \langle v_4 \rangle)$. In the first case, v_3 is placed in $p_{2,2,2}$ and v_4 is placed in $p_{2,3,3}$; we get $\Phi(x(e)) = \frac{2}{3}$ because it only spends two units (a unit represents the distance between two adjacent cells) of wirelength. In the second case, v_3 is placed in $p_{2,3,4}$ and v_4 is placed in $p_{3,2,4}$; we get $\Phi(x(e)) = \frac{2}{3}$ because it generates a power-consumption pair (v_3', v_4') and wirelength = 1. In the third case, v_3 is placed in $p_{2,3,2}$ and v_4 is placed in $p_{3,2,1}$; we get $\Phi(x(e)) = \frac{4}{3}$ (wirelength = 2; it contributes one memory space to $|m_{3,2}|$ and generates a power-consumption pair (v_3'', v_4'')).

4. OUR APPROACH

In this section, we present the algorithm for PCP. We consider partitioning and placement simultaneously in our method.

Figure 4 shows the framework for our placement algorithm. The first step is a precedence-constrained partitioning that partitions a circuit into r stages (associated with the CMPs) and minimizes the length of lifetimes of nodes, as the lengths of lifetimes affect the number of MRs needed for a DRFPGA. Moreover, a random initial solution may not satisfy the PCP because it may

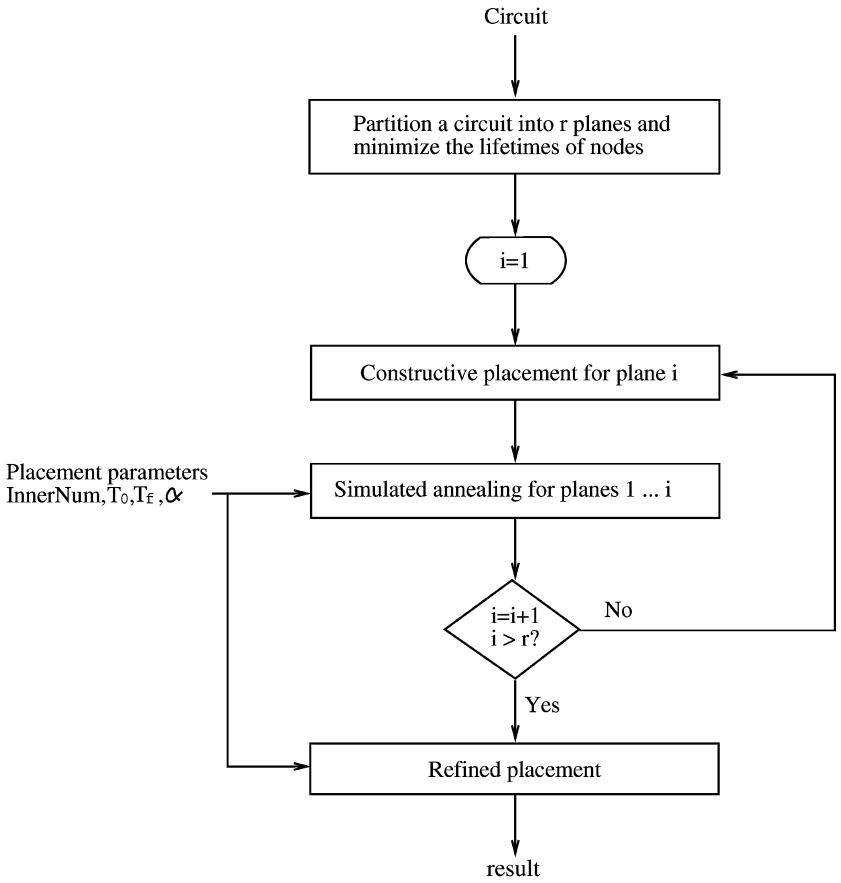

Fig. 4. High-level view of our placement algorithm.

violate precedence constraints. Our partitioning can handle the initialization of the PCP correctly. Once the partitioning is done, placement is performed for each CMP. We apply an iterative two-stage algorithm for each CMP i: an initial constructive method for CMP i followed by a simulated annealing method for CMPs 1 to i.

4.1 Precedence Constraint Partitioning

In the PCP, we first partition a circuit into r subcircuits and then apply the precedence-constrained placement algorithm to map each subcircuit to the corresponding CMP. In order to reduce the number of the MRs required for a circuit, we minimize the maximum density of live nodes. In this subsection, we propose an effective partitioning algorithm to shorten the lifetimes of nodes, which affect the number of MRs needed directly.

Our algorithm begins with an initial feasible partitioning which is usually the result of the ASAP and/or ALAP [Hitchcock and Thomas 1983] scheduling or is produced by using a constructive partitioning method. A node may be assigned to any CMP if the precedence constraints are not violated.

```
Algorithm: Partitioning_Algorithm_for_PCP
1  repeat
2      Free all tuples;
3      j = 1
4      while free_tuples ≠ ∅
5          select a tuple (i, k) with maximum g(i, k);
6          if it satisfies precedence and balance constraints
7              tentatively move the node v_i to the CMP k;
8              g_j = the gain due to the movement of v_i to the CMP k;
9              Update gains of all unlocked tuples;
10             j = j + 1;
11         Lock tuple (i, k);
12     endwhile
13     Find l that maximizes ĝ = Σ_{i=1}^{l} g_i;
14     if ĝ > 0
15         Make the moves of nodes belonging to a sequence from 1 to l permanent;
16 until ĝ ≤ 0
```

Fig. 5. Precedence-constrained partitioning algorithm for minimizing the lifetimes of nodes.

Given an initial partitioning, our algorithm improves the quality of the partitioning iteratively by selecting a set of tuples with the maximum accumulative gain, where the tuple is used to record the movement of nodes and is represented by *tuple*(*node, CMP*). *tuple*(v, i) is selected when the node v is moved into CMP i. Specifically, in an iteration, we select the *tuple*(v, i) which

1. has the maximum gain,
2. satisfies the precedence constraints, and
3. satisfies the balance criterion,

and a tentative move of the corresponding node is made. Then the gains associated with all neighbors of v are updated. A *tuple*(v, i) cannot be selected twice in an iteration. We repeat the selection process described above until all tuples are selected. In each iteration, all selected tuples and the resulting gains are recorded in order. The *partial sum* of the ith tuple is the total gains of the first i tuples. At the end of an iteration, the corresponding nodes of the maximum partial sum are moved. We then repeat the above action of an iteration until the maximum partial sum of an iteration is not greater than zero. This scheme is similar to the algorithm proposed by Fidducia and Mattheyses [1982]. The overall procedure of our partitioning method is described in Figure 5.

The gain function in our precedence-constrained partitioning is described in the following. The goal for the precedence-constrained partitioning is to minimize the maximum size of $cut(k)$, where $cut(k)$ denotes the set of MRs needed between CMP k and CMP $k+1$. If a node v_i is moved from CMP j to CMP k, then only the $cut(x)$, $\min\{j, k\} \leq x < \max\{j, k\}$ may be changed. Therefore, if node v is moved from CMP j to CMP k, the gain function $g_v(j, k)$ is given as follows.

$$g_v(j,k) = \max_{\min\{j,k\} \leq z < \max\{j,k\}} \{cut(z) \cdot \Delta_v(j,k)\}, \tag{2}$$

where $\Delta_v(j,k)$ denotes the change of the maximum $cut(z)$, $\min\{j,k\} \le z < \max\{j,k\}$; it is defined by

$$\Delta_v(j,k) = \max_{\min\{j,k\} \le z < \max\{j,k\}}\{cut_v(z)\} - \max_{\min\{j,k\} \le z < \max\{j,k\}}\{cut_{\overline{v}}(z)\},$$

where $cut_v(z)$ is the size of $cut(z)$ on the condition that v is still in CMP j, and $cut_{\bar{v}}$ is the size of $cut(z)$ on the condition that v is moved from CMP j to CMP k.

Before computing the gain values, we must check whether the movement of a node violates the precedence constraints. In each iteration, the gain value is calculated for each unlocked tuple.

4.2 Placement of Each CMP

After we have partitioned a circuit into r CMPs, a placement algorithm is then applied to the circuit. Our placement algorithm consists of two stages: the constructive method followed by the simulated annealing-based iterative improvement method.

4.2.1 *Constructive Placement.* To obtain a better initial placement, we use a constructive algorithm rather than a random one. Given a circuit $G(V,E)$, we first perform a random placement of all I/O nodes in the circuit, providing anchor points for the other unplaced nodes. For each unplaced node v, we define *p_degree(v)* and *np_degree(v)* as follows. *p_degree(v)* is the number of placed neighbors of v; in contrast, *np_degree(v)* is the number of unplaced neighbors of v. We set the priority of a node v depending on the ratio *p_degree(v)/(p_degree + np_degree(v))* (called the *p_ratio*). A node with higher p_ratio implies that most of its neighbors are placed; in other words, there is more precise information while it is placed. The placement ordering of nodes is done according to the priorities derived from their p_ratios. (We call this placement *p_ratio constructive placement*.) For a node v, we place v at the position associated with the arithmetic mean of all the placed neighbors of v. Due to the use of the neighborhood information from the placed nodes, our constructive method for PCPs leads to a superior initial placement for the simulated annealing method to that produced by a random method; as shown in Figure 6, the improvement is more significant when the size of the DRFPGAs is increasing.

4.2.2 *Simulated-Annealing-Based Iterative Improvement of Placement.* Following the constructive method for PCPs, a simulated annealing-based method [Kirkpatrick et al. 1983; Sechen and Sangiovanni-Vincentelli 1985] is applied to improve the initial placement. The simulated annealing method is one of the most well-developed and widely used iterative techniques for solving combinatorial optimization problems.

We perturb a feasible solution to another feasible solution by using the following moves.

M1: Swap two nodes of the same type in a CMP.

M2: Re-place a node in an empty cell with no precedence constraints.

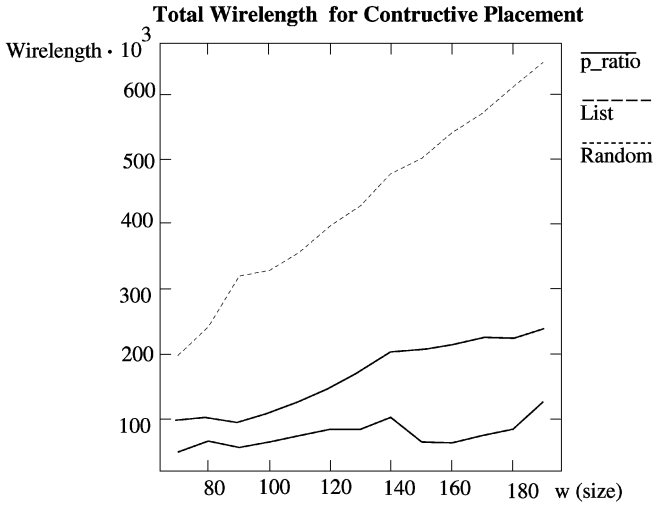

Fig. 6. Comparison of the wirelengths by using the p_ratio constructive placement, LIST, and randomly generated placement on DRFPGA of different sizes.

Table I. Characteristics of the ISCAS89 Benchmark Circuits in our Experiment

Circuit	#LUT	#DFF	#Nets	Depth
s5378	422	162	590	10
s9234	317	135	462	13
s13207	688	453	1121	14
s15850	1056	540	1570	23
s35932	2756	1728	4515	6
s38417	3458	1464	4894	18
s38584	3545	1294	4793	20

M3: Swap two nodes of the same type in the different CMPs with no precedence constraints.

5. EXPERIMENTAL RESULTS

Our algorithm has been implemented in the C++ language on a PC with a Pentium II 300 microprocessor and 512 MB RAM. We used the same test suite as Chang and Marek-Sadowska [1998], in which only the larger sequential circuits of the ISCAS'89 benchmarks were chosen. These circuits were translated by a technology mapper [Sentovich and Singh 1992] into 4-LUTs. Table I shows the characteristics of these benchmark circuits. Columns 2 through 4 list the number of LUT nodes, latch nodes, and nets in the circuits, respectively. In column 5, *depth* refers to the number of nodes on the critical path.

In our experiments, we estimated the wirelength by the *minimum spanning tree* method, which is very close to the actual routing length. We implemented the p_ratio constructive placement and compared the wirelength with List and random initialization on DRFPGAs of various sizes. The sizes of an array-based FPGA range from 70×70 to 200×200. As shown in Figure 6, the average wirelength of the initial placements generated by the p_ratio constructive

Table II. Results for the Eight-Stage DRFPGA Placement

Circuit	Random			List			Ours			
	Wirelength	$\|C\|$	$\text{Max}\{\|m_{i,j}\|\}$	Wirelength	$\|C\|$	$\text{Max}\{\|m_{i,j}\|\}$	Wirelength	$\|C\|$	$\text{Max}\{\|m_{i,j}\|\}$	Run-Time (s)
s5378	18129	1093	5	5655	1055	4	4315	549	3	551
s9234	15543	788	4	7953	763	4	5873	531	3	847
s13207	34562	1721	6	17014	1615	5	14208	908	3	1284
s15850	47948	2215	7	13360	2060	6	9873	1328	4	2048
s35932	105233	6657	6	33858	6073	6	27409	4267	5	3997
s38417	121954	8613	7	58048	8059	6	49026	5382	4	3871
s38584	139273	8711	8	60325	8066	6	51784	4795	5	4073
Total	482642	29798	43	196213	27691	37	162488	17760	27	16671

Table III. Minimizing $|C|$

Circuit	Minimal $\|C\|$			
	Wirelength	$\|C\|$	Max$\{\|m_{i,j}\|\}$	Run-Time (s)
s5378	4382	354	3	543
s9234	6023	319	3	901
s13207	15178	583	3	1136
s15850	10737	761	4	2109
s35932	29302	2173	5	3876
s38417	52004	3256	5	3836
s38584	53081	3628	5	4184
Total	170707	11074	28	16585

placement is substantially smaller than that generated by `List` and random initializations. In addition, the difference is more significant as the sizes of DRFPGAs increase. List scheduling labels each node with a priority and each node is greedily placed into a cell in an order according to the priorities. A placed node influences the priority of its neighbors. For the precedence constraints, most of the related research [Chang and Marek-Sadowska 1998, 1997; Trimberger 1998] applies the list scheduling heuristic with different priority models. But they only considered the partitioning problem. In our experiment, we adapted the list scheduling for the PCP. We compared our method with the list scheduling placement `List` and the randomly generated placement `Random` on the Xilinx DRFPGA model, in which a circuit was placed into 8 CMPs. The size of the DRFPGA was set to 25×25. The results are shown in Table II. Columns 2 to 4 list the total wirelength of `Random`, `List`, and our results (Ours), respectively. Columns 5 to 7 list the number of power-consumption pairs of `Random`, `List`, and Ours, respectively. Columns 8 to 10 list the maximum size of microregisters needed between two successive CMPs. The run-times of our algorithm are shown in Column 11. The last row in Table II reports the average improvement. The improvement for the `List`(`Random`) is calculated by (`List`(`Random`) $-$ *Ours*)/`List`(`Random`) $\times$ 100%. Overall, our method reduces the total wirelength, $|C|$, and max$\{|m_{i,j}|\}$ by 17.2 (66.3%), 35.9 (40.4%), and, 27% (37.2%), respectively, compared with `List` (`Random`).

In our next experiment, we try to minimize power-consumption in the PCP. We increase the rate of the penalty of power consumption. The results are shown in Table III. The total wirelengths, $|C|$, max$\{|m_{i,j}|\}$, and run-time are shown in Columns 2 to 5, respectively. The results show that the $|C|$ is reduced 37.6% if the wirelength is allowed to increase by 4.8%.

6. CONCLUSIONS

In this article, we have formulated a new precedence-constrained placement problem for Dynamically Reconfigurable FPGAs and presented an efficient algorithm to solve it. We proposed a new metric that can simultaneously consider wirelength, utilization of microregisters, and power consumption. We also presented a new constructive algorithm for initial placement. Experimental results show that, with the new metric, our algorithm outperforms `List` by a large margin.

REFERENCES

ALEXANDER, M., COHOON, J., GANLEY, J., AND ROBINS, G. 1995. Performance-oriented placement and routing for field-programmable gate arrays. In *Proceedings of the European Design Automation Conference (Brighton, UK, September 18–22)*, 80–85.

BHAT, N. B., CHAUDHARY, K., AND HAH, E. S. 1993. Performance-oriented fully routable dynamic architecture for a field programmable logic dervice. *Memorandum No. UCB/RELM93/42*, University of California, Berkeley.

BRGLEZ, F. 1993. ACM/SIGDA design automation benchmarks: Catalyst or anathema? *IEEE Des. Test*, Sept., 87–91.

BROWN, J., CHEN, D., TAU, E., ESLICK, I., AND DEHON, A. 1995. DELTA: Prototype for a first-generation dynamically programmable gate array. *Transit Note 112*, MIT, Cambridge, Mass.

CHANG, D. AND MAREK-SADOWSKA, M. 1997. Buffer minimization and time-multiplexed I/O on dynamically reconfigurable FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (Monterey, Calif., February)*, 142–148.

CHANG, D. AND MAREK-SADOWSKA, M. 1998. Partitioning sequential circuits on dynamically reconfigurable FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays, (Monterey, Calif., February)*, 161–167.

CHANG, Y.-W., THAKUR, S., ZHU, K., AND WONG, D. F. 1994. A new global routing algorithm for FPGAs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (San Jose, Calif., November 6–10)*, 380–385.

CHAO, C.-T. M., WU, G.-M., JIANG, H.-R., AND CHANG, Y.-W. 1999. A clustering- and probability-based approach for time-multiplexed FPGA partitioning. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided-Design (Las Vegas, November)*, 364–368.

CHEN, C. D., LEE, Y. S., WU, C. H., AND LIN, Y. L. 1995. TRACER-FPGA: A router for RAM-based FPGAs. *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 14*, 3 (Oct.), 371–374.

DEHON, A. 1994. DPGA-coupled microprocessors: Commodity ICs for the early 21st century. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, (Napa Valley, Calif., April 10–13)*, 31–39.

FIDDUCIA, C. M. AND MATTHEYSES, R. M. 1982. A linear-time heuristic for improving network partitions. In *Proceedings of the ACM/IEEE Design Automation Conference (Calif. June)*, 175–181.

HITCHCOCK III, C. Y. AND THOMAS, D. E. 1992. Iterative and adaptive slack allocation for performance-driven layout and FPGA routing. In *Proceedings of the ACM/IEEE Design Automation Conference (Anaheim, Calif. June 8–12)*, 536–542.

HITCHCOCK III, C. Y. AND THOMAS, D. E. 1983. A method of automatic data path synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference (Anaheim, Calif., June)*, 484–488.

JONES, D. AND LEWIS, D. M. 1995. A time-multiplexed FPGA architecture for logic emulation. In *Proceedings of the IEEE Custom Integrated Circuits Conference (Santa Clara, Calif., May 1–4)*, 495–498.

KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. 1983. Optimization by simulated annealing. *Science 220*, 4598, 671–680.

LEE, Y. S. AND WU, C. H. 1995. A performance and routability driven router for FPGAs considering path delay. In *Proceedings of the ACM/IEEE Design Automation Conference (Anaheim, Calif., June)* 557–561.

LIU, H. AND WONG, D. F. 1998. Network flow based circuit partitioning for time-multiplexed FPGAs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (San Jose, Calif., November)*, 497–504.

MCMURCHIE, L. AND EBELING, C. 1995. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (Monterey, Calif., February)*, 111–117.

SECHEN, C. AND SANGIOVANNI-VINCENTELLI, A. 1985. The TimberWolf placement and routing package. *IEEE J. Solid-State Circ. 20*, 2 (April), 510–522.

SENTOVICH, E. M. AND SINGH, K. J. 1992. SIS: A system for sequential circuit synthesis. Tech. Rep. M92/41, University of California, Berkeley.

TOGAWA, N., SATO, M., AND OHTSUKI, T. 1994. Maple: A simultaneous technology mapping placement, and global routing algorithm for field-programmable gate arrays. In *Proceedings of the*

IEEE/ACM International Conference on Computer-Aided Design (San Jose, Calif., November) 156–163.

TRIMBERGER, S. 1997. A time-multiplexed FPGA. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (Napa Valley, Calif., April 16–18)*, 22–28.

TRIMBERGER, S. 1998. Scheduling design into a time-multiplexed FPGA. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (Monterey, Calif., February)* 153–160.

WU, G.-M., LIN, J.-M., AND CHANG, Y.-W. 2001. Generic ILP-based approaches for time-multiplexed FPGA partitioning. *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 20*, 10 (Oct.), 1266–1274.

XILINX INC. 1996. *The Programmable Logic Data Book*. San Jose, Calif.

Received March 2001; revised April 2002; accepted September 2002