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 paper, 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: B7.1 [Integrated Circuits]: Types and Design Styles—gate arrays; B7.2 [Integrated Circuits]: Design Aids—placement and routing; J.6 [Computer Applications]: Computer-Aided Engineering

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

The work of Guang-Ming Wu was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-91-2215-E-333-001. The work of Yao-Wen Chang was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-91-2215-E-002-038.

Name: Guang-Ming Wu
Affiliation: Department of Information Management, Nan-Hua University, Chiayi, Taiwan. Email: gmwu@mail.nhu.edu.tw

Name: Jai-Ming Lin
Affiliation: Realtek Semiconductor Corp., No. 2, Industry E. Rd. IX, Science-Based Industrial Park, Hsinchu, Taiwan. Email: jarvis@realtek.com.tw

Name: Yao-Wen Chang
Affiliation: Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan. Email: ywchang@cc.ee.ntu.edu.tw

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.
1. INTRODUCTION

Improving logic efficiency by time-sharing, Dynamically Reconfigurable FPGAs (DRFPGGAs) have gained much attention recently. In a DRFGPA, a virtual large design is partitioned into multiple stages (or partitions) to share the same smaller physical device than that occupied by a traditional FPGA. Various architectures have been proposed, e.g., 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 programed 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 DRFGPA configuration model [Trimberger 1997]. The Xilinx DRFGPA 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 micro cycle, the SRAM bits of the corresponding configuration are loaded into the DRFGPA, and the configurable logic blocks (CLBs) are reused to evaluate combinational logic. One pass through all micro-cycles 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 micro registers (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 micro-cycles, but also hold latch values for use in the next user cycle. A micro-cycle starts with saving all the CLB results of the previous micro-cycle 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 is time-multiplexed on a DRFGPA, its 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 DRFGPAs. The DRFGPA partitioning problem has been studied in the recent literature [Chang and Marek-Sadowska 1997; Chang and Marek-Sadowska 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 micro-cycles or user cycles; nevertheless, the exact number of MRs needed will be known until placement stage. (It is possible to refine the partitioning results in the placement stage.) Moreover, there are some objectives must be considered in placement, like wirelength, which cannot be considered in partitioning. Therefore, the placement for DRFGPAs is important and valued problem that will be studied.

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 total wirelength that will be 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 the statement described in previous paragraphs, we need some MRs to store the values between micro-cycles (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. Besides, 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; Frankle 1992; Lee and Wu 1995; McMurchie and Ebeling 1995; Togawa et al. 1994]. The previous work applies some metrics to estimate wiring cost (wirlength, delay, etc) of a net; the metrics are usually incorporated in popular wiring estimation schemes such as semi-perimeter, Steiner tree, minimum spanning tree, etc. Due to the precedence and MR-capacity constraints, however, those metrics cannot apply to DRFPGA placement.

In this paper, our contributions, we introduce a new placement problem motivated by the DRFPGA architectures. For the DRFPGA placement, we develop in this paper 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 DRF-
PGAs. The first stage partitions a circuit into \( k \) sub-circuits 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 [Fiduccia and Mattheyses 1982] balanced bipartitioning. In the partitioning, we reduce the length of lifetime for each node as much as possible, since 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 that their neighbors are already placed. 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 with the list scheduling method.

The remainder of this paper 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 finally conclusions are given in Section 6.

2. PROBLEM FORMULATION

In this paper, 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 \( e = (v_1 \rightarrow v_2, v_3, \ldots, v_n) \), 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 \( E_e \) and \( E_f \) according to the type of fanout nodes. A net \( e \in E_e \) (\( E_f \)) 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 \( \text{Pre}(v) \) be the precedence of a node \( v \). For two nodes \( v \) and \( u \), we define \( \text{Pre}(v) \preceq \text{Pre}(u) \) 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 \( \text{Pre}(v) \preceq \text{Pre}(u) \). 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, i.e., the data of the node must be stored in an MR for later use. For the fanout node $v_1$ of a net $e = \langle v_1 \rightarrow v_2, v_3, \ldots, v_n \rangle$, and if $e \in E_c$, the lifetime of $v_1$ is from the CMP $s(v_1)$ to the CMP $\max\{s(v_j) \mid 2 \leq j \leq n\}$, 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 $\max\{s(v_j) \mid 2 \leq j \leq n\}$, because the output of a latch node is used 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 $v_1$ does 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 lifetime 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 lifetime overlap, as shown in Figure 2(b). Therefore, the lifetimes of nodes affect the number of MRs required.

![Fig. 2. (a) Since the lifetimes of related nodes $v_1$ and $v_2$ do not overlap, and thus their results can store in the same MR. (b) The lifetimes of related nodes $v_3$ and $v_4$ overlap, their results must be stored in different MRs.](image-url)

In Xilinx DRFPGAs, during reconfiguration stage, first, the logic and interconnect array are updated simultaneously from a configuration memory plane. Secondly, it must to settle the signals from the micro-registers. 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 $Pre(u) \geq Pre(v)$ 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, e.g., the case of nodes $v_1$ and $v_2$ in Figure 3. If $Pre(u) \leq Pre(v)$ 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.

![Diagram](image)

Fig. 3. Three case of placing net \((v_3 \rightarrow v_4)\) to empty cells, with the other net \((v_1 \rightarrow v_2)\) being preplaced. Case 1: \(v_3\) and \(v_4\) are placed in the same CMP, only wiring length must be considered in metric. Case 2: \(v'_3\) and \(v'_4\) have wirelength and power consumption penalties, it has no wiring and power-consumption penalty. Case 3: \(v''_3\) and \(v''_4\), they have wirelength, memory and power penalties.

We use the following notations throughout this paper.

- \(c(u, v)\): A power-consumption pair for nodes \(u\) and \(v\).
- \(C\): The set of all power-consumption pairs in the placement.
- \(|C|\): The number of power-consumption pairs in \(C\).
- \(B = (P, M)\): \(P\) is the set of configuration 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 \(k\).
- \(P = \bigcup_{k \in \{1,2,...,r\}} P_k\), where \(r\) is the number of stages (CMCs) 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 \(b_{i,j}\).
- \(p_{k,i,j}\): An LUT cell in \(b_{i,j}\).
- \(p'_{k,i,j}\): 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 \(p'_{k,i,j}\) \((p'_{k,i,j})\), 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) \( \| \mathcal{C} \| \)
are simultaneously minimized, and for any nodes \( v_1 \) and \( v_2 \), \( s(v_1) \leq s(v_2) \) if \( \text{Pre}(v_1) \leq \text{Pre}(v_2) \).

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, 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 paper, we are first 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 the three issues: (1) wirelength, (2) micro register requirement, (3) power consumption. The metric presented in this paper is defined as follows. Let \( x(e) \) be the placement of a net \( e \) which satisfies the precedence constraints. The cost for \( x(e) \), \( \Phi(x(e)) \), is given by

\[
\Phi(x(e)) = \alpha w(x(e)) + \beta h(x(e)) + \gamma o(x(e)),
\]

where \( w(x(e)) \), \( h(x(e)) \), and \( o(x(e)) \) represent the respective cost functions for wirelength, MR count, and power consumption, and \( \alpha, \beta, \gamma \) 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 \( (v_1 \rightarrow v_2, v_2 \rightarrow v_3 \rightarrow v_4) \) has been preplaced and \( |m_{i,j}| \) 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 \( e = (v_3 \rightarrow v_4) \). In the first case, \( v_3 \) is placed in \( p_{2,2} \) and \( v_4 \) is placed in \( p_{2,3,3} \); we get \( \Phi(x(e)) = \frac{2}{3} \) since 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 \( \Phi(x(e)) = \frac{2}{3} \) since 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 \( \Phi(x(e)) = \frac{3}{3} \) (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 (asso-
associated with the CMPs) and minimizes the length of lifetimes of nodes, since the lengths of lifetimes affect the number of MRs needed for a DRFPGA. Moreover, a random initial solution may not satisfy the PCP since it may violate precedence constraints. Our partitioning can handle the initial 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 $i$.

![Diagram](image)

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

### 4.1 Precedence Constraint Partitioning

In the PCP, we first partition a circuit into $r$ sub-circuits and then apply the precedence-constrained placement algorithm to map each sub-circuit to the corre-
sponding 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.

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 move of nodes and is represented by \( \text{tuple}(node, \text{CMP}) \). \( \text{tuple}(v, i) \) is selected when the node \( v \) is moved into CMP \( i \). Specifically, in an iteration, we select the \( \text{tuple}(v, i) \) 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 \( \text{tuple}(v, i) \) 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 \text{partial sum} of the \( i \)th 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. 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 Fiduccia and Mattheyses [Fiduccia 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 \( \text{cut}(k) \), where \( \text{cut}(k) \) 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 \( \text{cut}(x) \), \( \min\{j, k\} \leq x < \max\{j, k\} \) 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:

\[
g_v(j, k) = \max_{\min\{j, k\} \leq z < \max\{j, k\}} \{ \text{cut}(z) \cdot \Delta_v(j, k) \},
\]

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

\[
\Delta_v(j, k) = \max_{\min\{j, k\} \leq z < \max\{j, k\}} \{ \text{cut}_v(z) \} - \max_{\min\{j, k\} \leq z < \max\{j, k\}} \{ \text{cut}_\pi(z) \},
\]

where \( \text{cut}_v(z) \) is the size of \( \text{cut}(z) \) on condition that \( v \) is still in CMP \( j \), and \( \text{cut}_\pi \) is the size of \( \text{cut}(z) \) on condition that \( v \) is moved from CMP \( j \) to CMP \( k \).

Before computing the gain values, we must check whether the move of a node violates the precedence constraints. In each iteration, the gain value is calculated.
Algorithm: Partitioning Algorithm for \( J \)PCP

1. repeat
2. Free all tuples;
3. \( j = 1 \)
4. while free tuples \( \neq \emptyset \)
5. select a tuple \( (i, k) \) with maximum \( g(i, k) \);
6. if it satisfies precedence and balance constraints
7. tentative move the node \( v_i \) to the CMP \( k \);
8. \( g_j = \text{the gain due to the move of } v_i \text{ to the CMP } k \);
9. Update gains of all unlocked tuples;
10. \( j = j + 1 \);
11. Lock tuple \( (i, k) \);
12. endwhile
13. Find \( t \) that maximizes \( \hat{g} = \sum_{i=1}^{t} g_i \);
14. if \( \hat{g} > 0 \)
15. Make the moves of nodes belonging to a sequence from 1 to \( t \) permanent;
16. until \( \hat{g} \leq 0 \)

for each unlocked tuple.

4.2 Placement of Each CMPs

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. It provides anchor points for the other unplaced nodes. For each unplaced node \( v \), we define \( p_{\text{degree}}(v) \) and \( n p_{\text{degree}}(v) \) as follows. \( p_{\text{degree}}(v) \) is the number of placed neighbors of \( v \); in contrast, \( n p_{\text{degree}}(v) \) is the number of unplaced neighbors of \( v \). We set the priority of a node \( v \) depending on the ratio \( \frac{p_{\text{degree}}(v)}{p_{\text{degree}}(v) + n p_{\text{degree}}(v)} \) (called \( p_{\text{ratio}} \)). A node with higher \( p_{\text{ratio}} \) 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 according to the priorities derived from their \( p_{\text{ratios}} \). (We called this placement as \( p_{\text{ratio}} \) 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 than that produced by a random method; as shown in Figure 6, the improvement is more significant when the size of 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 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 three moves.

M1: Swap two nodes of the same type in a CMP.
M2: Replace a node to an empty cell with no precedence constraints.
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 use the same test suite as [Chang and Marek-Sadowska 1998], in which only the larger sequential circuits of the ISCAS’89 benchmarks are chosen. These circuits are translated by technology mapper [Sentovich and Singh 1992] into 4-LUTs. Table 1 shows the characteristic of these benchmark circuits. Columns 2, 3 and 4 list the number of LUTs nodes, latch nodes and nets in the circuits, respectively. In column 5, depth refers to the number of nodes on the critical path.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#LUT</th>
<th>#DFF</th>
<th>#Nets</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3378</td>
<td>422</td>
<td>162</td>
<td>590</td>
<td>10</td>
</tr>
<tr>
<td>s0234</td>
<td>317</td>
<td>135</td>
<td>462</td>
<td>13</td>
</tr>
<tr>
<td>s33207</td>
<td>688</td>
<td>453</td>
<td>1121</td>
<td>14</td>
</tr>
<tr>
<td>s35850</td>
<td>1056</td>
<td>540</td>
<td>3570</td>
<td>23</td>
</tr>
<tr>
<td>s35032</td>
<td>2756</td>
<td>1728</td>
<td>4515</td>
<td>6</td>
</tr>
<tr>
<td>s38417</td>
<td>3458</td>
<td>1464</td>
<td>4894</td>
<td>18</td>
</tr>
<tr>
<td>s82854</td>
<td>3545</td>
<td>1294</td>
<td>4793</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of the ISCAS89 benchmark circuits in our experiment.

In our experiments, we estimate the wirelength by the minimum spanning tree method, which is very closely to the actual routing length. We implemented the $p_{ratio}$ constructive placement and compared the wirelength with List and random initialization on DRFGAs of various sizes. The sizes of an array-based FPGA range from 70 x 70 to 200 x 200. As shown in Figure 6, the average wirelength of the initial placements generated by the $p_{ratio}$ constructive placement is substantially smaller than those generated by List and random initializations. In addition, the difference is more significant as the sizes of DRFGAs 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; Chang and Marek-Sadowska 1997; Trimberger 1998] applies the list scheduling heuristic with different priority models. But they only considered the partitioning problem. In our experiment, we adapt 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 DRFGA model, in which a circuit was placed into 8 CMPs. The size of the DRFGA is set to $25 \times 25$. The
results are shown in Table 2. Columns 2–4 list the total wirelength of Random, List and our results (ours), respectively. Columns 5–7 list the number of power-consumption pairs of Random, List and Ours, respectively. Columns 8–10 list the maximum size of micro registers needed between two successive CMPs. The runtimes of our algorithm are shown in Column 11. The last row in Table 2 reports the average improvement. The improvement for the List(Random) is calculated by \( \frac{\text{List(Random)}}{\text{List(Random) - Ours}} \times 100\% \). Overall, our method reduces the total wirelength, \( |C| \) and \( \max\{m_{ij}\} \) 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 3. The total wirelengths, \( |C|, \max\{|m_{ij}|\} \) and runtime are shown in Columns 2–5, respectively. The result show that the \( |C| \) is reduced 37.6\% if the wirelength is allowed to increase by 4.8\%.

6. CONCLUSIONS

In this paper, 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 which can simultaneously consider wirelength, utilization of micro registers, 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.
Table 2: Results for the 8-stage DIFPGA placement.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Alpha</th>
<th>Beta</th>
<th>Gamma</th>
<th>Delta</th>
<th>Eta</th>
<th>Zeta</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time (s)</td>
<td>1.5</td>
<td>2.0</td>
<td>2.5</td>
<td>3.0</td>
<td>3.5</td>
<td>4.0</td>
<td>4.5</td>
</tr>
<tr>
<td>Vertical</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>Horizontal</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>22</td>
<td>26</td>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>Wirelength</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
</tbody>
</table>

| Performance-driven Placement for Dynamically Reconfigurable FPGAs | 13 |
\begin{table}
\centering
\begin{tabular}{|l|c|c|c|c|}
\hline
Circuit & \multicolumn{3}{c|}{minimal $|C|$} & Run Time [s] \\
\hline
 & Wirelength & $|C|$ & $\max(\{|C|\})$ & \\
\hline
n5378 & 4382 & 301 & 3 & 543 \\
n6231 & 6023 & 319 & 3 & 901 \\
n12327 & 15178 & 563 & 3 & 1.366 \\
n13580 & 10747 & 501 & 4 & 2.286 \\
n35062 & 20602 & 2173 & 5 & 3.878 \\
n38417 & 52004 & 3256 & 5 & 3.878 \\
n38584 & 53081 & 3628 & 5 & 4.842 \\
Total & 170007 & 14074 & 28 & 36585 \\
\hline
\end{tabular}
\caption{Minimizing $|C|$.}
\end{table}

REFERENCES


Xilinx Inc. 1996. The Programmable Logic Data Book. San Jose, California.

Received April 2001; revised April 2002; accepted September 2002