Generic ILP-based Approaches for Dynamically Reconfigurable FPGA Partitioning

Guang-Ming Wu[1], Jai-Ming Lin[2], Mango C.-T. Chao[2], and Yao-Wen Chang[3]

[1]Department of Information Management, Nan-Hua University, Chiayi, Taiwan
gmwu@mail.nhu.edu.tw
[2]Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan
{gis87808, gis87530}@cis.nctu.edu.tw
[3]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
ywchang@cc.ee.ntu.edu.tw

Abstract

Due to the precedence constraints among vertices, the partitioning problem for dynamically reconfigurable FPGAs (DRFPGAs) is different from the traditional one. In this paper, we first derive logic formulations for the precedence-constrained partitioning problems, and then transform the formulations into integer linear programs (ILPs). The ILPs can handle the precedence constraints and minimize cut sizes simultaneously. To enhance performance, we also propose a clustering method to reduce the problem size. Experimental results based on the Xilinx DRFPGA architecture show that our approach outperforms the list scheduling (`List`), the network flow based (`FBB-m`) [14], and the probability based (`PAT`) [6] methods by respective average improvements of 46.6%, 32.3%, and 21.5% in cut sizes. Our approach is practical and scales well to larger problems; the empirical runtime grows close to linearly in the circuit size. More importantly, our approach is very flexible and can readily extend to the partitioning problems with various objectives and constraints, which makes the ILP formulations superior alternatives to the DRFPGA partitioning problems.

1 Introduction

Dynamically reconfigurable FPGAs (DRFPGAs) improve logic efficiency by dynamically re-using hardware. Currently there is fast growing interest in DRFPGAs for reconfigurable computing. In DRFPGAs, a large design can be partitioned into multiple stages to share the same smaller physical device in different time frames. Several different architectures have been proposed, e.g., the Xilinx architecture [16], the Virtual Element Gate Array [12], the Dynamically Programmable Gate Array [3, 7], Dharma [1], etc. All these models allow dynamic reuse of logic blocks and wire segments by reprogramming on-chip SRAM bits.

Figure 1 shows the Xilinx DRFPGA configuration model [16]. The DRFPGA emulates a single large design through multiple configurations. Circuit configuration can be partitioned into multiple stages and stored in the *configuration memory planes* (*CMPs*). The DRFPGA can hold only one active configuration in any time frame. Each configuration is called a *micro-cycle*, and one pass through all micro-cycles is called a *user cycle*. All combinational logic is evaluated, and flip-flop values are updated in one user cycle. The target architecture consists of an array of augmented XC4000-style Control-Logic-Blocks (CLBs) [16, 17]. Each CLB includes *micro registers* (*MRs*) to store the intermediate values of combinational logic for use in later micro-cycles and also hold the flip-flop values for use in the next user cycle. A micro-cycle starts with saving all CLB results of the previous micro-cycle in MRs, and then a new configuration is loaded into the active configuration memory.

Because the logic and interconnect needed for a circuit is dynamically reconfigurable on a DRFPGA, its partitioning problem is 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 [4, 5, 6, 14, 16]. Chang and Marek-Sadowska in [4] and [5] presented the list-scheduling methods (`List`) for buffer register and cut-size minimization for various DRFPGA architectures. Liu and Wong in [14] proposed a network-flow-based algorithm (`FBP-m`) for multi-stage precedence-constrained partitioning for the Xilinx-like DRFPGAs. Recently, Chao et al. in [6] proposed a probability-based approach (`PAT`) for the partitioning problem on the Xilinx-like DRFPGAs. The probability-based approach combines second-order information and a stochastic-gain function [9], the FM partitioning-based iterative-improvement method [10], and the MFFC [8] based clustering. It gives the best results among the previous works for the DRFPGA partitioning problem.

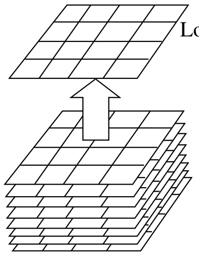


Figure 1: The Xilinx dynamically reconfigurable FPGA configuration model.

In this paper, we present generic integer linear programming (ILP) formulations for the multi-stage precedence-constrained partitioning problems. We begin with a mathematical description of the partitioning objectives and constraints, which can be translated into integer linear programs. Unlike most existing methods that can consider the precedence constraints and cut sizes only in some local stages at a time, the ILP-based method can consider those for all stages simultaneously and thus has a more global perspective to optimize given objectives. To enhance performance, we also propose a clustering method to reduce the problem size; the clustering provides a trade-off between runtime and solution quality (in terms of CLB and interconnection costs). Experimental results based on the Xilinx DRFPGA architecture show that our approach outperforms the list-scheduling `list`, the network-flow-based (FBP-m)[14], and the probability-based (PAT) [6] by respective average improvements of 46.6%, 32.3%, and 21.5% in cut sizes. More importantly, our algorithm is very practical and scales well to larger problems; the empirical runtime grows close to linearly in the circuit size. Its runtimes range from 38 minutes for the smallest circuit (s820) to about 6 hours for the largest circuit (s35932). Moreover, our approach is very flexible and can readily extend to the partitioning problems with various objectives and constraints, e.g., buffer-register minimization [4], etc. The flexibility makes the ILP formulations superior alternatives to the DRFPGA partitioning.

The remainder of this paper is organized as follows. Section 2 formulates the DRFPGA partitioning problem. Section 3 presents the ILP formulations for the problem. Section 4 proposes a clustering method to enhance runtime. Section 5 extends our approach to the DRFPGA partitioning problems with various objectives and constraints. Section 6 reports the experimental results. Finally, conclusions are given in Section 7.

2 Problem Formulation

A circuit can be represented by a graph $G = (V, E)$. Each vertex $v \in V$ (edge $e \in E$) corresponds to a gate (net) in the circuit. The vertices in V consist of *combinational vertices* and *flip-flop vertices*. A combinational (flip-flop) vertex represents a combinational gate (flip-flop) in a circuit. Each vertex $v \in V$ has a weight $w(v)$. The weight of a subset V' of V, denoted by $w(V')$, is given by $\sum_{v \in V'} w(v)$.

For a net e with n vertices $v_1, v_2, \ldots, v_n$, let v_1 be the *fanout* vertex whose output signal is the input signal to v_j $(2 \leq j \leq n)$, and v_j is the *output* vertex of v_1. The nets E can be divided into two categories E_c and E_f according to the type of fanout vertices. A net $e \in E_c$ (E_f) if the fanout vertex v_1 is a combinational (flip-flop) vertex. A net is called a *two-terminal* net if it connects only two vertices; if it connects more than two vertices, then it is a *multi-terminal* net. We denote a net e by $e = (v_1 \rightarrow < v_2, v_3, \ldots, v_n >)$ where v_1 is the fanout vertex and $v_2, v_3, \ldots, v_n$ are its output vertices.

For a DRFPGA, a circuit is partitioned into several stages such that the logic in different stages temporally share the same physical FPGA CLBs. Each stage forms a *micro cycle*, and one pass of all stages forms a *user cycle*. MRs store the intermediate values of combinational logic for use in later micro cycles in the same user cycle and store flip-flop values for use in the next user cycle. Therefore, the interconnections among stages determine the number of MRs that the DRFPGA needs. We label interconnections between stages i and $i+1$ as I_i and denotes the size of I_i as $|I_i|$. We call I_i as the i-th *cut* representing the cut (interconnections) between stages i and $i+1$, $1 \leq i \leq p-1$, where p is the number of memory planes (MPs) in a DRFPGA. Moreover, I_p represents the cut between stage p in this user cycle and stage 1 in the next user cycle. According to the Xilinx-like architecture [16], the following precedence constraints must be satisfied: (1) Each combinational vertex must be scheduled in a stage no later than all its output vertices. (2) Each flip-flop vertex must be scheduled in a stage no earlier than all its input and output vertices. These constraints define a partial temporal ordering on the vertices in the circuit. Let $Pre(v)$ be the precedence of a vertex v. For two vertices v_1 and v_2, we define $Pre(v_1) \preceq Pre(v_2)$ if v_1 must be scheduled no later than v_2. In other words, in order to produce the correct result in one user cycle for a partitioned dynamically reconfigurable circuit, the virtual CLBs must be evaluated in the proper order (see Figure 2), and the MRs used in a micro cycle cannot exceed the number of actual CLBs in a DRFPGA. We call this type of partitioning as *precedence-constrained partitioning*.

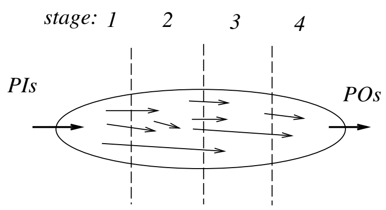


Figure 2: Four stages in the precedence-constrained partitioning of a circuit.

By the above constraints, we formulate the *memory plane (MP) constrained partitioning* problem for DRFPGAs as follows:

- **The MP-Constrained Partitioning (MPCP) Problem**.
  Input: Given a circuit $G = (V, E)$ and the number of memory planes p in a DRFPGA.
  Problem: Determine the p-stage precedence-constrained partitioning with the following objectives:

  1. *Balance objective*: Minimize $\max\{w(V_i) | 1 \leq i \leq p\}$, where V_i denotes the vertices in the stage i.
  2. *Min-cut objective*: Minimize $\max\{|I_i| \, | 1 \leq i \leq p\}$.

The logic in different stages share the same physical CLBs in a dynamically reconfigurable manner. Hence, the CLBs used in a micro cycle cannot exceed the number of actual CLBs in a DRFPGA, i.e., the number of vertices should be smaller than the number of actual CLBs. The balance objective, minimize $max\{w(V_i)|1 \leq i \leq p\}$, is to balance the sizes of stages so that the design can fit into a smaller physical device. The min-cut objective minimizes the maximum cuts between successive stages.

3 ILP Formulations for the MPCP Problem

In this section, we first describe the ILP formulations for the MPCP problem. To reduce the total execution time of a user cycle, we add the ILP formulations for timing associated with the *Temporal Precedence Graph* (*TPG*).

3.1 Two-Terminal Nets

The notations used in our formulations are defined as follows. Suppose that a hyper-graph $G(V, E)$ with n vertices and m nets is partitioned into p stages.

In the MPCP problem, given a circuit $G(V, E)$ and a DRFPGA with p memory planes, the circuit is partitioned into p stages without violating the precedence constraints, balance objective, and the cost of the partitioning is minimized, where the cost consists of the maximum numbers of interconnections and CLBs needed in a stage. The above cost can be minimized by the ILP formulations presented in the following.

The variables used in the formulations are as follows.

- M_c: An integer variable that denotes the number of CLBs needed in the DRFPGA.
- M_r: An integer variable that denotes the number of micro registers (or interconnections) needed in the DRFPGA.
- $x_{i,j}$: A 0-1 integer variable associated with v_i. $x_{i,j} = 1$ if v_i is assigned to the stage j; otherwise, $x_{i,j} = 0$.
- $y_{i,k}$: A 0-1 integer variable associated with a net $e_i = (v_{i_1} \rightarrow < v_{i_2} >)$, where $e_i \in E$. $y_{i,k} = 1$ if net $e_i = (v_{i_1} \rightarrow < v_{i_2} >)$ introduces an interconnection between the stages k and $k+1$ (kth cut); otherwise, $y_{i,k} = 0$.

Liu and Wong in [14] used an *α-bounded uni-directional min-cut* to ensure that M_c is bounded in the range $[(1-\epsilon)\alpha, (1+\epsilon)\alpha]$, e.g., $\epsilon = 5\%$ in their implementation. In this section, we follow the α-bounded approach to limit the maximum number of CLBs needed in each stage. Thus, we only consider the maximum number of interconnections needed in each stage as the cost function. For the p-stage precedence-constrained partitioning, $\alpha = \frac{|V|}{p}$.

For a net $e_i \in E_c$, $y_{i,k} = 1$ if v_{i_1} is assigned to stage $1, \ldots,$ or k and v_{i_2} is assigned to stage $k+1, \ldots,$ or p; otherwise, $y_{i,k} = 0$. Thus, $y_{i,k}$ is given as follows:

$$\begin{aligned} y_{i,k} = \quad & (\mathbf{OR}(x_{i_1,1}, \ldots, x_{i_1,k})) \ \mathbf{AND} \\ & (\mathbf{OR}(x_{i_2,k+1}, \ldots, x_{i_2,p})), 1 \leq k \leq p-1. \end{aligned} \tag{1}$$

By the precedence constraint, v_{i_1} must be assigned to a stage no later than v_{i_2}. Therefore, the two **OR** terms in the above equation cannot be zero at the same time. Equation (1) is mathematically equivalent to Equation (2):

$$y_{i,k} = \left(\sum_{s=1}^{k} x_{i_1,s} + \sum_{s=k+1}^{p} x_{i_2,s} \right) - 1, \ 1 \leq k \leq p-1. \tag{2}$$

For a net $e_i \in E_c$, $y_{i,p}$ is always equal to zero, since the data of a combinational node is only used in the current user cycle.

For a net $e_i = (v_{i_1} \rightarrow < v_{i_2} >) \in E_f$, $y_{i,k} = 1$ if v_{i_1} and v_{i_2} are both assigned to stages $1, \ldots, k$, or v_{i_1} and v_{i_2} are both assigned to stages $k+1, \ldots, p$; otherwise, $y_{i,k} = 0$. (See Figure 3 for a graphical representation.)

We formulate $y_{i,k}$ as follows:

$$\begin{aligned} y_{i,k} = \quad & ((\mathbf{OR}(x_{i_1,1}, \ldots, x_{i_1,k}))\mathbf{AND}(\mathbf{OR}(x_{i_2,1}, \ldots, x_{i_2,k}))) \\ \mathbf{OR} \quad & ((\mathbf{OR}(x_{i_1,k+1}, \ldots, x_{i_1,p}))\mathbf{AND}(\mathbf{OR}(x_{i_2,k+1}, \ldots, x_{i_2,p}))), \\ & 1 \leq k \leq p-1 \end{aligned} \tag{3}$$

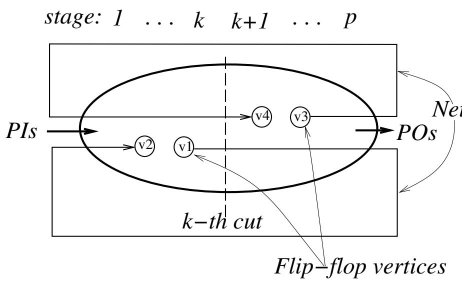


Figure 3: Two flip-flop nets $(v_1 \rightarrow< v_2 >)$ and $(v_3 \rightarrow< v_4 >)$. Each of them contributes an interconnection to the k-th cut.

Similarly, Equation (3) is mathematically equivalent to Equation (4) shown below.

$$y_{i,k} = \quad 2 - \left(\sum_{s=1}^{k} x_{i_2,s} + \sum_{s=k+1}^{p} x_{i_1,s} \right) \quad 1 \leq k \leq p-1. \tag{4}$$

For $e_i \in E_f$, $y_{i,p}$ is always equal to one, since the value from a flip-flop vertex must be stored in a micro register for later use in the next user cycle.

Therefore the MPCP problem can be formulated as follows.

$$\begin{aligned}
&Minimize && M_r \\
&Subject\ to && (1-\epsilon)\alpha \leq M_c \leq (1+\epsilon)\alpha && (5)\\
& && \sum_{i=1}^{n} x_{i,j} - M_c \leq 0, 1 \leq j \leq p && (6)\\
& && \sum_{j=1}^{p} x_{i,j} = 1, 1 \leq i \leq n && (7)\\
& && \sum_{k=1}^{p} kx_{i_1,k} - \sum_{k=1}^{p} kx_{i_2,k} \leq 0, \\
& && \text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >), e_i \in E_c && (8)\\
& && \sum_{k=1}^{p} kx_{i_2,k} - \sum_{k=1}^{p} kx_{i_1,k} \leq 0, \\
& && \text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >), e_i \in E_f && (9)\\
& && \sum_{e_i \in E} y_{i,k} - M_r \leq 0, 1 \leq k \leq p. && (10)
\end{aligned}$$

The objective function is used to minimize the number of micro registers. Constraint (5) ensures that M_c deviates within the range $[(1-\epsilon)\alpha, (1+\epsilon)\alpha]$. Constraint (6) states that each stage cannot contain more than M_c CLBs. It is clear that $v_i \in V$ can be assigned in exactly one stage, which is formulated in Constraint (7). Constraints (8) and (9) ensure that the precedence relations of the graph will be preserved. For any net $e_i = (v_{i_1} \rightarrow< v_{i_2} >)$, if $e_i \in E_c$ (E_f), then Constraint (8) (Constraint (9)) ensures that v_{i_1} will be scheduled in a stage no later (earlier) than v_{i_2}. Constraint (10) states that any cut size between two adjacent stages cannot exceed M_r—an interconnection in a cut needs a micro register to save its signal.

3.2 Multi-Terminal Nets

In this section, we present the ILP formulation for multi-terminal nets. We redefine the 0-1 integer variable $y_{i,k}$ associated with a multi-terminal net $e_i = (v_i \rightarrow< v_{j_1}, v_{j_2}, \ldots, v_{j_q} >)$. $y_{i,k} = 1$ if the net introduces an interconnection between stages k and $k+1$; otherwise, $y_{i,k} = 0$.

First, we consider multi-terminal combinational nets in a circuit. For a net $e_i = (v_i \rightarrow< v_{j_1}, v_{j_2}, \ldots, v_{j_q} >)$ in E_c, v_i is a combinational vertex and $Pre(v_i) \preceq Pre(v_{j_t})$ for $1 \leq t \leq q$. v_i must be in a stage no later than its output vertices $v_{j_t}{}'s$ $(1 \leq t \leq q)$. Therefore, we shall rewrite Constraint (8) for net e_i as follows.

$$\sum_{k=1}^{p} kx_{i,k} - \sum_{k=1}^{p} kx_{j_t,k} \leq 0, \quad 1 \leq t \leq q.$$

For a net $e_i \in E_c$, $y_{i,k} = 1$ if v_i is assigned to stage $1, 2, \ldots$, or k and any of $v_{j_1}, \ldots, v_{j_q}$ is assigned to stage $k+1, \ldots$, or p; otherwise, $y_{i,k} = 0$. Thus, $y_{i,k}$ is given as follows:

$$\begin{aligned}
y_{i,k} = \quad & (\mathbf{OR}(x_{i,1}, \ldots, x_{i,k})) \ \mathbf{AND} \ (\mathbf{OR}(x_{j_1,k+1}, \ldots, x_{j_1,p})) \\
\mathbf{OR} \quad & (\mathbf{OR}(x_{i,1}, \ldots, x_{i,k})) \ \mathbf{AND} \ (\mathbf{OR}(x_{j_2,k+1}, \ldots, x_{j_2,p})) \\
\mathbf{OR} \quad & \ldots \\
\mathbf{OR} \quad & (\mathbf{OR}(x_{i,1}, \ldots, x_{i,k})) \ \mathbf{AND} \ (\mathbf{OR}(x_{j_q,k+1}, \ldots, x_{j_q,p})).
\end{aligned} \tag{11}$$

Let

$$\begin{aligned}
z_{i_t,k} = \quad & (\mathbf{OR}(x_{i,1}, \ldots, x_{i,k})) \ \mathbf{AND} \ (\mathbf{OR}(x_{j_t,k+1}, \ldots, x_{j_t,p})), \\
& 1 \leq t \leq q.
\end{aligned} \tag{12}$$

Rewriting Equation (11), we have

$$y_{i,k} = \quad z_{i_1,k} \ \mathbf{OR} \ z_{i_2,k} \ \mathbf{OR} \quad \ldots \ \mathbf{OR} \ z_{i_q,k}.$$

Similar to Equation (2), we have

$$z_{i_t,k} = \quad \sum_{s=1}^{k} x_{i,s} + \sum_{s=k+1}^{p} x_{j_t,s} - 1, \ 1 \leq t \leq q. \tag{13}$$

In addition, we need the following constraint to satisfy the definition of $y_{i,k}$.

$$\sum_{t=1}^{q} z_{i_t,k} - qy_{i,k} \leq 0, \ 1 \leq k \leq p-1. \tag{14}$$

The reason for Constraint (14) is given as follows. The variables $z_{i_t,k} (1 \leq t \leq q)$ and $y_{i,k}$ are 0-1 integers. When any of $z_{i_t,k}{}'s (1 \leq t \leq q)$ is set to 1, it also makes $y_{i,k}$ equal 1. When all $z_{i_t,k}{}'s$ are set to 0, $y_{i,k}$ can be set to 0 or 1. By Constraint (10), we have $\sum_{e_i \in E} y_{i,k} \leq M_r$. Since the objective function will minimize M_r, $y_{i,k}$ will be set to 0 when all $z_{i_t,k}{}'s$ are set to 0. Therefore, Equation (13) is mathematically equivalent to Equation (14).

We now consider multi-terminal flip-flop nets. For a net $e_i = (v_i \rightarrow< v_{j_1}, v_{j_2}, \ldots, v_{j_q} >)$ in E_f, v_i is a flip-flop vertex and $Pre(v_{j_t}) \preceq Pre(v_i)$ for $1 \leq t \leq q$. v_i must be in a stage no earlier than its output vertices $v_{j_t}{}'s$ $(1 \leq t \leq q)$. Therefore, we can rewrite Constraint (9) for the net e_i as follows.

$$\sum_{k=1}^{p} kx_{j_t,k} - \sum_{k=1}^{p} kx_{i,k} \leq 0, \quad 1 \leq t \leq q.$$

For a net $e_i \in E_f$, $y_{i,k} = 1$ if v_i and any of $v_{j_t}{}'s$ (where $1 \leq t \leq q$) are both assigned to stages $1, \ldots, k$ or stages $k+1, \ldots, p$; otherwise, $y_{i,k} = 0$. We formulate $y_{i,k}$ as follows:

$$\begin{aligned}
y_{i,k} = \quad & (((\mathbf{OR}(x_{i,1}, \ldots, x_{i,k}))\mathbf{AND}(\mathbf{OR}(x_{j_1,1}, \ldots, x_{j_1,k})))\mathbf{OR} \\
& ((\mathbf{OR}(x_{i,k+1}, \ldots, x_{i,p}))\mathbf{AND}(\mathbf{OR}(x_{j_1,k+1}, \ldots, x_{j_1,p})))) \\
\mathbf{OR} \quad & (((\mathbf{OR}(x_{i,1}, \ldots, x_{i,k}))\mathbf{AND}(\mathbf{OR}(x_{j_2,1}, \ldots, x_{j_2,k})))\mathbf{OR} \\
& ((\mathbf{OR}(x_{i,k+1}, \ldots, x_{i,p}))\mathbf{AND}(\mathbf{OR}(x_{j_2,k+1}, \ldots, x_{j_2,p})))) \\
\mathbf{OR} \quad & \ldots \\
\mathbf{OR} \quad & (((\mathbf{OR}(x_{i,1}, \ldots, x_{i,k}))\mathbf{AND}(\mathbf{OR}(x_{j_q,1}, \ldots, x_{j_q,k})))\mathbf{OR} \\
& ((\mathbf{OR}(x_{i,k+1}, \ldots, x_{i,p}))\mathbf{AND}(\mathbf{OR}(x_{j_q,k+1}, \ldots, x_{j_q,p})))), \\
& 1 \leq k \leq p-1.
\end{aligned}$$

Let

$$z'_{i_t,k} = (((\mathbf{OR}(x_{i,1},\ldots,x_{i,k}))\mathbf{AND}(\mathbf{OR}(x_{j_t,1},\ldots,x_{j_t,k})))\mathbf{OR}$$
$$((\mathbf{OR}(x_{i,k+1},\ldots,x_{i,p}))\mathbf{AND}(\mathbf{OR}(x_{j_t,k+1},\ldots,x_{j_t,p})))), 1 \le t \le q.$$

Rewriting Equation (15), we have

$$y_{i,k} = \quad z'_{i_1,k}\ \mathbf{OR}\ z'_{i_2,k}\ \mathbf{OR} \quad \ldots\ \mathbf{OR}\ z'_{i_q,k},\ 1 \le k \le p-1.$$

By Equation (4), we have

$$z'_{i_t,k} = \quad 2 - \left(\sum_{s=1}^{k} x_{j_t,s} + \sum_{s=k+1}^{p} x_{i,s}\right),\ 1 \le t \le q. \tag{15}$$

Similar to Constraint (14), we need the following constraint to satisfy the definition of $y_{i,k}$:

$$\sum_{r=1}^{q} z'_{i_r,k} - q y_{i,k} \le 0,\ 1 \le k \le p-1. \tag{16}$$

Now we consider the p-th cut. For any flip-flop net $e_i = (v_i \to< v_{j_1}, v_{j_2}, \ldots, v_{j_n} >)$, $y_{i,p}$ always equals to 1 because the output of v_i must be saved for use in the next user cycle. Thus, we have the following constraint for each flip-flop net:

$$y_{i,p} = 1.$$

3.3 Performance-Driven MPCP (PDMPCP) Problem

The critical path in each plane determines the execution time of a DRFPGA. For a circuit partitioned into p stages, we set an upper bound D for the length of the critical path, *depth*, of each stage. In our method, $D = \lceil \frac{depth}{p} \rceil$, where *depth* is the critical path of the circuit. We call the partitioning with a bound of the critical path as *bounded-delay precedence-constrained partitioning*.

The ASAP and ALAP [11] scheduling algorithms are often used to identify the movable range of a vertex. ASAP (ALAP) determines the earliest (latest) possible stage of a vertex. For each vertex v, let $A_S(v)$ and $A_L(v)$ be the stage assigned to v in the ASAP and ALAP scheduling, respectively. If $A_S(v) = A_L(v)$, then vertex v must be fixed in the stage $A_S(v)$. Otherwise, a vertex v can be assigned to the stages in $[A_S(v), A_L(v)]$.

Considering the movable range estimated by the ASAP and ALAP scheduling, we reformulate Constraints (7)–(9) as follows:

$$\sum_{j=A_S(v_i)}^{A_L(v_i)} x_{i,j} = 1, 1 \le i \le n; \tag{17}$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} k x_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} k x_{i_2,k} \le 0,$$
$$\text{for each } e_i = (v_{i_1} \to< v_{i_2} >),\ e_i \in E_c; \tag{18}$$

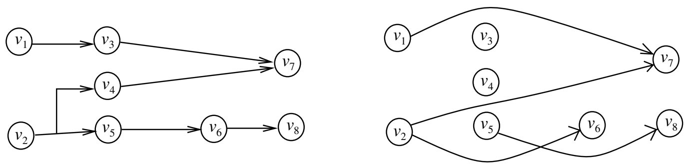
$$\sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} k x_{i_2,k} - \sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} k x_{i_1,k} \le 0,$$
$$\text{for each } e_i = (v_{i_1} \to< v_{i_2} >),\ e_i \in E_f. \tag{19}$$

However, $A_S(v)$ and $A_L(v)$ only give the lower and upper bounds of the movable range for partitioning vertex v. With the delay-bound constraint, the movable range for the vertex v may be tighter than $[A_S(v), A_L(v)]$. Thus, we have the following lemma.

Lemma 1 *The movable range of a vertex v, $[A_S(v), A_L(v)]$, derived from the ASAP and ALAP scheduling is a necessary but not sufficient condition for v to be feasible in the bounded-delay precedence-constrained partitioning.*

In the following, we present the *Temporal Precedence Graph* (*TPG*) and translate the constraints associated with TPG (*TPG constraints*) into ILP formulations, which is a precise limitation for the relative positions of nodes.

A pair of vertices (v_i, v_j) is *critical* if the length of the longest path between v_i and v_j is equal to $D + 1$. (We do not need to consider those pairs of vertices with their longest paths greater than $D + 1$.) Thus, v_i and v_j cannot be assigned to the same stage if the pair (v_i, v_j) is critical. Given a circuit $G = (V, E)$, its corresponding TPG $G_t = (V_t, E_t)$ is constructed by $V_t = \{V_c | V_c \in V\}$ and $E_t = \{(v_i \to< v_j >) |$ pair (v_i, v_j) is critical$\}$. Figure 4 shows a TPG example which contains eight vertices $v_1, v_2, \ldots, v_8$. Assume that the delay bound in a stage is equal to two (i.e., $D = 2$). Then, two vertices cannot be assigned to the same stage if there exists a path between them and they are not consecutive. As an example in Figure 4(a), the length of the longest path associated with (v_1, v_7) is equal to 3; therefore, they cannot be assigned to the same stage, and we connect (v_1, v_7) with a directed edge (see Figure 4(b)). Similarly, there are directed edges (v_2, v_7), (v_2, v_6), and (v_5, v_8) in the TPG shown in Figure 4.



(a) (b)

Figure 4: (a) A piece of combinational part of a given circuit. (b) The corresponding TPG with $D = 2$.

For a sequential circuit, $depth$ is the length of the longest path of the combinational part. Therefore, for a sequential circuit $G(V, E)$, we can remove all nets in E_f and all flip-flop vertices, and obtain a directed acyclic graph $G_a(V_a, E_a)$. The TPG $G_t(V_t, E_t)$ can then be constructed from G_a by the similar method as discussed earlier. If there is no path between two vertices in G_a, there is no precedence relation between them. Thus, it suffices to consider every pair of connected vertices in G_a. For each vertex v_i, we add the edge $(v_i \to< v_j >)$ to E_t if the length of longest path between v_i and v_j (a successor of v_i) equals $D+1$. Given a circuit $G(V, E)$, Algorithm TPG_Generation shown in Figure 5 generates a TPG associated with $G(V, E)$.

Algorithm: TPG_Generation($G(V,E), p$)
Input: $G(V,E)$—a circuit graph;
p—the number of memory planes in a DRFPGA.
Output: $G_t(V_t, E_t)$—TPG of G;
$V = V - V_f; E \leftarrow E - E_f;$
$V_t = V; E_t = \emptyset;$
$D = \lceil \frac{depth}{p} \rceil;$
for each vertex $v_i \in V$
  for each immediate successor v_j of v_i in G
    Timing_Constraint_Edge(1, i, j);
Output $G_t(V_t, E_t)$.

Subroutine: $Timing\_Constraint\_Edge(path\_length, f, t)$
if $path\_length + 1 > D$ **then**
  $E_t = E_t \cup \{(v_f, v_t)\};$
else
  for each immediate successor v_k of v_t in G
    Timing_Constraint_Edge($path\_length + 1$, f, k);

Figure 5: The algorithm for generating TPG.

Two vertices in a critical pair cannot be assigned to the same stage. Therefore, we can incorporate the critical pairs into the MPCP formulation to ensure that the execution time of every stage do not exceed D.

The constraint can be formulated as follows:

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \leq -1,$$
$$\text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >),\ e_i \in E_t. \quad (20)$$

Our ILP formulation is summarized in Figure 6, and Theorem 1 states the correctness of the formulation.

Theorem 1 *The problem $MPCP$ has a solution if and only if all vertices V in G can be partitioned into p stages in the DRFPGA under the precedence and delay-bound constraints.*

Problem **PDMPCP**
Minimize M_r
Subject to

$$(1-\epsilon)\alpha \leq M_c \leq (1+\epsilon)\alpha \quad (21)$$

$$\sum_{i=1}^{n} x_{i,j} - M_c \leq 0, 1 \leq j \leq p. \quad (22)$$

$$\sum_{j=A_S(v_i)}^{A_L(v_i)} x_{i,j} = 1, 1 \leq i \leq n. \quad (23)$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \leq 0,$$
$$\text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >),\ e_i \in E_c. \quad (24)$$

$$\sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} - \sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} \leq 0,$$
$$\text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >),\ e_i \in E_f. \quad (25)$$

$$\sum_{t=1}^{q} z_{i_t,k} - qy_{i,k} \leq 0, 1 \leq k \leq p-1, \text{for each } e_i \in E_c. \quad (26)$$

$$\sum_{r=1}^{q} z'_{i_r,k} - qy_{i,k} \leq 0, 1 \leq k \leq p-1, \text{for each } e_i \in E_f. \quad (27)$$

$$\sum_{e_i \in E} y_{i,k} - M_r \leq 0, 1 \leq k \leq p. \quad (28)$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \leq -1,$$
$$\text{for each } e_i = (v_{i_1} \rightarrow< v_{i_2} >),\ e_i \in E_t. \quad (29)$$

Figure 6: The ILP formulation for PDMPCP.

3.4 Complexity of the PDMPCP Problem

The complexity of the PDMPCP problem can be analyzed in terms of the numbers of variables and constraints. In the PDMPCP problem, the number of stages is given and the values of the 0-1 variables $y_{i_1,i_2}{}'s$ can be obtained by $x_{j_1,j_2}{}'s$. Thus the only unknowns are 0-1 variables $x_{i,j}{}'s$, M_c, and M_r. The number of $x_{i,j}{}'s$ is given by $\Sigma_{i=1}^{n}(A_L(v_i) - A_S(v_i)+1)$ which is bounded by pn, where p is the number of stages and n is the number of vertices in the circuit. Note that, by Constraint (23), only n variables can be set to 1. Thus once $x_{i,j}$ is set to 1 , the remaining variables associated with the range $[A_L(v_i), A_S(v_i)]$ are implicitly set to 0. Therefore, pn is in fact a very loose upper bound for the number of variables for $x_{i,j}{}'s$.

We analyze the number of equations needed for a circuit in the following. It is obvious that the number of equations required for Constraints (21), (22), (23), and (28) is 1, p, n, and p respectively. For each two-terminal (multi-terminal) net, we need an equation for Constraint (24) or (25) ((26) or (27)) depending on the type of the net, a combinational net or a flip-flop net. Therefore, the total number of equations required is m' (m), where m' (m) is the number of two-terminal (multi-terminal) nets in a circuit. We can add one edge at most for each pair of vertices in a TPG; therefore, the number of equations required for Constraint (29) is n^2. According to these analyses, the total number of equations required is $O(n^2 + m')$.

4 Solution Space Reduction

An effective clustering algorithm can greatly improve the quality of the precedence-constrained partitioning and speed up the partitioning algorithm by reducing the problem size. Sanker and Rose [15] proposed a new clustering metric that is effective in clustering traditional circuits, but may not generate feasible clusters due to the precedence constraints in the DRFPGA partitioning.

Based on Sanker and Rose's algorithm [15], we propose in the following a clustering method which can consider the precedence constraints during clustering. Our clustering algorithm begins by randomly choosing some vertices as seeds. Each unclustered vertex connected to those seeds is assigned scores used to decide to which cluster the vertex belong. The score, $w_{v,c}$, for each candidate vertex v associated with a cluster c has two components:

- *The number of connections between the candidate vertex v and the cluster c being considered, with each connection weighted by the fanout of the net on which it lies.*
- *The number of nets that would be completely absorbed if this candidate vertex v were added to the cluster c.*

A net is said to be *absorbed* by a cluster if all the vertices on that net are contained within that single cluster. Let $N_{v,c}$ denote the set of nets shared between the candidate vertex v and the cluster c, P_v the set of pins on net $e \in N_{v,c}$, and $A_{v,c}$ the set of nets absorbed by adding the candidate vertex v to the cluster c, then the score can be expressed as

$$w_{v,c} = \sum_{e \in N_{v,c}} \frac{1}{|P_v| - 1} + |A_{v,c}|.$$

With this function, vertices on low-fanout nets and on nets that are about to be absorbed are preferred when building the clusters. For a candidate vertex we pick the highest score associated with a cluster and add the vertex to the cluster. This process is repeated until all vertices are clustered. The result is a netlist of clusters with absorbed nets being removed.

However, the above procedure might not satisfy the precedence constraints. It may generate cycles in the graph. To ensure that no cycle is generated, we cluster according to the topological order of vertices in the circuit. Moreover, we check a circuit whether there is any cycle created when clustering. The algorithm is named *Precedence-Constrained Clustering* and is summarized in Figure 7. Theorem 2 gives the time complexity of the algorithm.

Theorem 2 *Algorithm Precedence-Constrained Clustering runs in $O(n^2 + m)$ time, where n (m) is the number of vertices (nets) in the circuit.*

5 Extensions

Our approach is very flexible; it can also handle the precedence-constrained partitioning of different forms with only minor modifications. In this section, we extend the ILP formulations to the buffer-register minimization partitioning problem that was first investigated in [4]. The buffer-register minimization partitioning problem is addressed on the Dharma [1] architecture. As mentioned in [4], buffer registers

Algorithm: $Precedence\_Constrained\_Clustering(G, k)$
Input: $G(V, E)$—a circuit;
k—the number of clusters.
Output: $S = \{C_1, C_2, \ldots, C_k\}$—a set of clusters.
1. Randomly pick k vertices $v_1, v_2, \ldots, v_k$ as seeds;
2. Let T is a set of the other unclustered vertices;
3. **while** T is non-empty **do**
4. Remove a vertex v connected to a cluster C_i from T;
5. Compute the set of scores $W = \{w_{v,C_1}, w_{v,C_2}, \ldots, w_{v,C_k}\}$;
6. Pick the cluster C_i associated with the maximum of W;
7. **if** $Merge(v_i, v)$ does not create a cycle in G **then**
8. $C_i \leftarrow C_i \cup \{v\}$;
9. $Merge(v_i, v)$;
10. **else**
11. Remove the maximum from $\{w_{v,C_1}, w_{v,C_2}, \ldots, w_{v,C_k}\}$;
12. goto line 6;
13. **Output** S.

Subroutine: $Merge(v_i, v)$
1. **for** each edge e_i between v_i and v **do**
2. $E = E - e_i$;
3. **for** each edge $e_j = (v \rightarrow < v_j >)$ **do** /* $v_j \in V$ */
4. $E = E \cup \{(v_i, v_j)\}$;
5. **for** each edge $e'_j = (v_j \rightarrow < v >)$ **do** /* $v_j \in V$ */
6. $E = E \cup \{(v_j, v_i)\}$;
7. $V = V - \{v\}$.

Figure 7: The clustering algorithm.

are needed because the dynamically reconfigurable nature of DRFPGAs means that only a portion of the circuit implemented on the chip is present at any given time instance. Thus there is a need to buffer signal until they are no longer needed. Figure 8 shows the partitioning with different buffer register requirements. Buffer registers are used to store signals among nonadjacent stages. Three buffer registers are needed in the partitioning shown in Figure 8(a) while only two buffer registers are required in that shown in Figure 8(b). We denote the set of buffer registers needed in the stage i as B_i and the total number of buffer registers in B_i as $|B_i|$. Based on the model in [4], $|B_i| = |\{e|$ the fanout vertex v of net e is assigned earlier than the stage i, and one of $v's$ outputs is assigned later than the stage i $\}|$. We formulate the Buffer-Register Minimization Partitioning Problem as follows:

- **The Buffer-Register Minimization Partitioning (BRMP) Problem**.
  Input: Given a circuit $G = (V, E)$ and the number of memory planes p in a DRFPGA.
  Problem: Determine the p-stage precedence-constrained partitioning with the following objective:
  Minimize $M_c + \alpha M_b$,
  where α is a user specified parameter.

We give the formulation for two-terminal nets as follows. Extensions to multi-terminal nets are similar to the techniques presented in Section 3.2. Let M_b be a new integer variable that denotes the number of buffer registers needed in the DRFPGA, $z''_{i,k}$ a 0-1 integer variable associated with $e_i = (v_i \rightarrow < v_j >)$. $z''_{i,k} = 1$ if we need a buffer register in stage k to store a signal between v_i and v_j; otherwise, $z''_{i,k} = 0$. We formulate $z''_{i,k}$ as follows:

$$z''_{i,k} = (\mathbf{OR}(x_{i,1}, x_{i,2}, \ldots, x_{i,k-1})) \;\mathbf{AND}\; (\mathbf{OR}(x_{j,k+1}, x_{j,k+2}, \ldots, x_{j,p})), \text{ for each } e_i \in E_c.$$

Applying similar techniques for $y_{i,k}$ as described in Section 3.1, we obtain the formulation for the BRMP problem as follows.

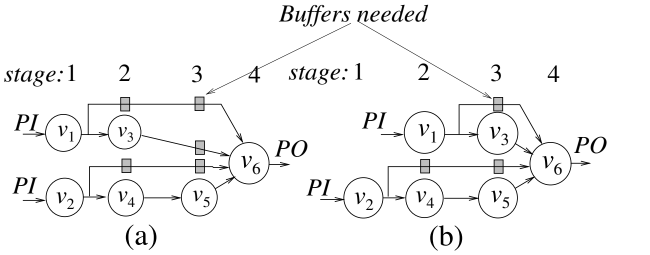


Figure 8: Buffer registers are needed to store signals among nonadjacent stages. (a) Three buffer registers are needed since $\max\{|B_i||1 \le i \le 4\} = 3$ ($|B_3| = 3$). (b) Only two buffer registers are needed since $\max\{|B_i||1 \le i \le 4\} = 2$ ($|B_3| = 2$).

Minimize $M_c + \alpha M_b$
Subject to

$$(1-\epsilon)\alpha \le M_c \le (1+\epsilon)\alpha \quad (30)$$

$$\sum_{i=1}^{n} x_{i,j} - M_c \le 0, 1 \le j \le p \quad (31)$$

$$\sum_{j=A_S(v_i)}^{A_L(v_i)} x_{i,j} = 1, 1 \le i \le n \quad (32)$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \le 0, \text{ for each } e_i = (v_{i_1} \rightarrow < v_{i_2} >),\ e_i \in E_c \quad (33)$$

$$\sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} - \sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} \le 0, \text{ for each } e_i = (v_{i_1} \rightarrow < v_{i_2} >),\ e_i \in E_f \quad (34)$$

$$\sum_{t=1}^{q} z_{i_t,k} - qy_{i,k} \le 0, 1 \le k \le p-1, \text{for each } e_i \in E_c. \quad (35)$$

$$\sum_{r=1}^{q} z'_{i_r,k} - qy_{i,k} \le 0, 1 \le k \le p-1, \text{for each } e_i \in E_f. \quad (36)$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \le -1, \text{ for each } e_i = (v_{i_1} \rightarrow < v_{i_2} >),\ e_i \in E_t \quad (37)$$

$$\sum_{e_i \in E} z''_{i,k} \le M_b, 2 \le k \le p-1, \quad (38)$$

where α is user specified parameter, and $\alpha \ge 0$. The new constraint (38) states that each stage cannot contain more than M_b buffer registers.

6 Experimental Results

The programs for our system (the ASAP and ALAP scheduling—i.e., bounded-delay precedence-constrained partitioning presented in Section 3.3, clustering, and ILPs) were written in the C++ language, and the ILPs were solved using the LINDO package [13] on a PC with a Pentium II 300 microprocessor and 512 MB RAM. LINDO starts with a feasible linear programming solution and searches for optimal integer solutions using the branch-and-bound method. To speed up the runtime, we search at most five feasible solutions in using LINDO. We tested on the MCNC Partitioning93 benchmark circuits [2] used in [4, 5, 6, 14].

We compared our method with the list scheduling `List` [4, 5], the network flow based approach `FBP-m` [14] and the probability based approach `PAT` [6] on the Xilinx DRFPGA model, in which a circuit was

Circuit	Max # of registers				Ours Imprv. (%)		
	List	FBP-m (sec.)	PAT (sec.)	Ours (sec.)	List	FBP-m	PAT
c3540	177	166 (0.31)	126 (1)	135 (3558)	+23.7	+18.7	−7.1
c5315	265	165 (0.78)	157 (7)	106 (4856)	+60.0	+35.8	+32.5
c6288	117	114 (0.13)	114 (2)	92 (6114)	+21.4	+19.3	+19.3
c7552	453	392 (0.54)	260 (17)	204 (5648)	+55.0	+48.0	+21.5
s820	91	81 (0.01)	43 (1)	45 (2287)	+50.5	+44.4	−4.7
s838	131	71 (0.05)	72 (1)	37 (2677)	+71.8	+47.9	+48.6
s1423	130	120 (0.10)	106 (1)	56 (3126)	+56.9	+53.3	+47.2
s9234	640	502 (2.09)	430 (26)	331 (7641)	+48.3	+34.1	+23.0
s13207	1118	901 (5.09)	838 (180)	725 (12133)	+35.2	+19.5	+13.5
s15850	1070	877 (3.97)	808 (106)	696 (14265)	+35.0	+20.6	+13.9
s35932	3806	2950 (19.14)	2138 (21106)	2026 (21491)	+46.8	+31.3	+5.2
s38417	3546	2892 (218.45)	2628 (1067)	1655 (20510)	+53.3	+42.8	+37.0
s38584	5131	2796 (84.86)	3611 (2397)	2673 (21188)	+47.9	+4.4	+30.0
Average					+46.6	+32.3	+21.5

Table 1: Results for the 8-stage DRFPGA partitioning.

partitioned into eight stages. The size of a stage is bounded by the balance factor 5%. This is the same as in [5, 6, 14]. The results are shown in Table 1. Columns 2, 3 and 4 in Table 1 list the maximum numbers of micro registers (cuts) used by `List`, `FBP-m` and `PAT`, respectively. Column 5 lists the maximum numbers of micro registers used by our algorithm, and the runtimes are shown in brackets. Columns 6, 7, and 8 list the percentages of improvements of ours over `List`, `FBP-m`, and `PAT`, respectively. The improvement for the `List` (`FPB-m, PAT`) is calculated by $\frac{\texttt{List (FPB-m, PAT)} - Ours}{\texttt{List (FPB-m, PAT)}} \times 100\%$. The results show that our method on the average reduces the maximum numbers of micro registers required by 46.6%, 32.3%, and 21.5%, compared with `List`, `FBP-m` and `PAT`, respectively. The results show the effectiveness of our ILP approach. In particular, our approach is practical and scales well to larger problems. As shown in Figure 9, the runtime is plotted as a function of the circuit size, the empirical runtime grows close to linearly in the circuit size. The runtimes depend on the numbers of 0-1 variables and range from 38 minutes for the smallest circuit s820 to about 6 hours for the largest circuit s35932. (In the ILP formulation, the numbers of variables and constraints needed by the largest circuit s35932 are 11328 and 5046, respectively.) Note that the runtimes include those for all the C++ programs (the ASAP and ALAP scheduling, clustering, and the ILP equation generation) and the LINDO program. (For the circuit s35932, `PAT` (on an Intel Pentium II 300 PC) requires about 5.5 hours, and `List` (on an Ultra Sparc 1 workstation) consumes about $1.5-$ hours on a different DRFPGA architecture.)

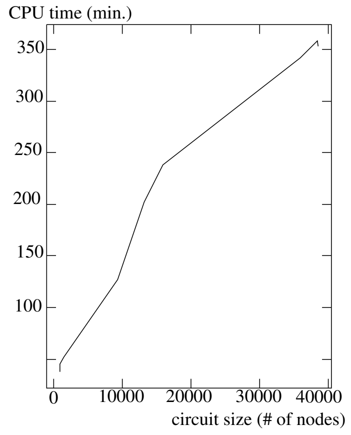


Figure 9: The runtime requirement vs. circuit size.

7 Conclusion

We have presented generic ILP formulations for a *set* of multi-stage precedence-constrained partitioning problems and a clustering method for reducing problem sizes. Experimental results have shown the effectiveness of the ILP-based approaches. The ILP-based formulations are so flexible that they can readily apply to the partitioning problems with various objectives and constraints. The flexibility makes the ILP formulations superior alternatives to the DRFPGA partitioning.

8 Acknowledgments

This work was partially supported by National Science Council of Taiwan ROC under Grant No. NSC-89-2215-E-009-054. The authors would like to thank Dr. Huiqun Liu for the benchmark circuits and Prof. Wai Kei Mak for his constructive comments.

References

[1] N. B. Bhat, et al., "Performance-oriented fully routable dynamic architecture for a field programmable logic device," Memmorandum No. UCB/RELM93/42, University of California, Berkeley, 1993.

[2] F. Brglez, "ACM/SIGDA Design Automation Benchmarks: Catalyst or Anathema?," *IEEE Design and Test*, pp. 87–91, Sep., 1993.

[3] J. Brown, et al. "DELTA: Prototype for a first-generation dynamically programmable gate array," Transit Note 112, MIT, 1995.

[4] D. Chang and M. Marek-Sadowska, "Buffer minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs," *Proc. FPGA'97*.

[5] D. Chang, et al, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," *Proc. FPGA'98*.

[6] M. Chao, et al., "A Clustering- and Probability-Based Approach for Time-Multiplexed FPGA Partitioning," *Proc. ICCAD'99*.

[7] A. DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century," *Proc. FCCM'94*.

[8] J. Cong, et al, "Acyclic Multi-Way Partitioning of Boolean Networks," *Proc. DAC'94*.

[9] S. Dutt and W. Deng, "Partitioning Using Second-Order Information and Stochastic-Gain Functions," *Proc. ISPD'98*.

[10] C. M. Fidducia, et al, "A linear-time heuristic for improving network partitions", *Proc. DAC'82*.

[11] C. Y. Hitchcock III, et al, "A Method of Automatic Data Path Synthesis," *Proc. DAC'83*.

[12] D. Jones and D. M. Lewis, "A time-multiplexed FPGA architecture for logic emulation," *Proc. CICC'95*.

[13] LINDO: Linear INteractive and Discrete Optimizer for linear, integer, and quadratic programming problems LINDO Systems, Inc., 1999.

[14] H. Liu and D. F. Wong, "Network Flow Based Circuit Partitioning for Time-multiplexed FPGAs," *Proc. ICCAD'98*.

[15] Y. Sankar and J. Rose, "Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs," *Proc. FPGA'99*.

[16] S. Trimberger, "A Time-Multiplexed FPGA," *Proc. FCCM'97*.

[17] Xilinx, "The Programmable Logic Data Book," 1996.