Generic ILP-Based Approaches for Time-Multiplexed FPGA Partitioning

Guang-Ming Wu, Jai-Ming Lin, and Yao-Wen Chang

***Abstract*—Due to the precedence constraints among vertices, the partitioning problem for time-multiplexed field-programmable gate arrays (TMFPGAs) 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 TMFPGA architecture show that our approach outperforms the list-scheduling (List), the network-flow-based (FBB-m) (Liu and Wong, 1998), and the probability-based (PAT) (Chao, 1999) 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 TMFPGA partitioning problems.**

***Index Terms*—Layout, partitioning, physical design.**

I. Introduction

Time-multiplexed field-programmable gate arrays (TMFPGAs) improve logic efficiency by dynamically re-using hardware. Currently there is fast growing interest in TMFPGAs for reconfigurable computing. In TMFPGAs, 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 [17], the virtual element gate array [13], 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 static random access memory (SRAM) bits.

Fig. 1 shows the Xilinx TMFPGA configuration model [17]. The TMFPGA 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 TMFPGA can hold only one active configuration in any time frame. Each configuration is called a *microcycle* and one pass through all microcycles 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) [17], [18]. Each CLB includes microregisters (MRs) to store the intermediate values of combinational logic for use in later microcycles and also hold the flip-flop values for use in the next user cycle. A microcycle starts with saving all CLB results of the previous microcycle in MRs and then a new configuration is loaded into the active configuration memory.

Manuscript received June 1, 2000; revised February 26, 2001. This work was supported in part by the National Science Council of Taiwan, R.O.C., under Grant NSC-89-2215-E-009-054. This paper was recommended by Associate Editor C.-K. Cheng.

G.-M. Wu is with the Department of Information Management, Nan-Hua University, Dalin Chiayi 622, Taiwan (e-mail: gmwu@mail.nhu.edu.tw).

J.-M. Lin is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: gis87808@cis.nctu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan (e-mail: ywchang@cc.ee.ntu.edu.tw).

Publisher Item Identifier S 0278-0070(01)08887-X.

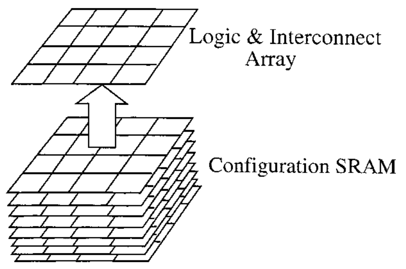


Fig. 1. Xilinx TMFPGA configuration model.

Because the logic and interconnect needed for a circuit is time-multiplexed on a TMFPGA, its partitioning problem is different from traditional ones. (This partitioning is similar to the scheduling problem in high-level synthesis [12].) The major difference is that the execution order of circuit elements must follow the precedence constraints in the TMFPGAs. The TMFPGA partitioning problem has been studied in the recent literature [4]–[6], [15], [17]. Chang and Marek-Sadowska in [4] and [5] presented the list-scheduling methods (List) for buffer-register and cut-size minimization for various TMFPGA architectures. Liu and Wong in [15] proposed a network-flow-based algorithm (FBP-m) for multistage precedence-constrained partitioning for the Xilinx-like TMFPGAs. Recently, Chao *et al.* in [6] proposed a probability-based approach (PAT) for the partitioning for the Xilinx-like TMFPGAs. The probability-based approach combines second-order information and a stochastic-gain function [9], the Fiduccia and Mattheyses partitioning-based iterative-improvement method [10], and the maximum fan-out-free cone-based clustering [8]. It gives the best results among the previous works for the TMFPGA partitioning problem.

In this paper, we present generic integer linear programming (ILP) formulations for the multistage precedence-constrained partitioning problems. We begin with a mathematical description of the partitioning objectives and constraints, which can easily 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 tradeoff between runtime and solution quality (in terms of CLB and interconnection costs). Experimental results based on the Xilinx TMFPGA architecture show that our approach outperforms the list-scheduling (List), the network-flow-based (FBP-m) [15], 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 min for the smallest circuit (s820) to about 6 h 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]. The flexibility makes the ILP formulations superior alternatives to the TMFPGA partitioning.

The remainder of this paper is organized as follows. Section II formulates the TMFPGA partitioning problem. Section III presents the ILP formulations for the problem. Section IV proposes a clustering method to enhance runtime. Section V extends our approach to the TMFPGA partitioning problems with various objectives and

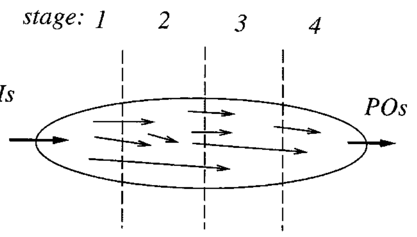


Fig. 2. Four stages in the precedence-constrained partitioning of a circuit.

constraints. Section VI reports the experimental results. Finally, conclusions are given in Section VII.

II. 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 *fan-out* vertex whose output signal is the input signal to v_j $(2 \le j \le n)$ and v_j is the *output* vertex of v_1. The nets E can be divided in two categories: E_c and E_f according to the type of fan-out vertices. A net $e \in E_c$ (E_f) if the fan-out 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 *multiterminal* net. We denote a net e by $e = (v_1 \rightarrow \langle v_2, v_3, \ldots, v_n \rangle)$, where v_1 is the fan-out vertex and $v_2, v_3, \ldots, v_n$ are its output vertices.

For a TMFPGA, 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 *microcycle* and one pass of all stages forms a *user cycle*. MRs store the intermediate values of combinational logic for use in later microcyles 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 TMFPGA 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 ith *cut* representing the cuts (interconnections) between stages i and $i+1$, $1 \le i \le p-1$, where p is the number of memory planes (MPs) in a TMFPGA. Moreover, I_p represents the cuts between stage p in this user cycle and stage 1 in the next user cycle. According to the Xilinx-like architecture [17], the following precedence constraints must be satisfied: 1) each combinational vertex must be scheduled in a stage no later than all its output vertices and 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 $\mathrm{Pre}(v)$ be the precedence of a vertex v. For two vertices v_1 and v_2, we define $\mathrm{Pre}(v_1) \preceq \mathrm{Pre}(v_2)$ 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 time-multiplexed circuit, the virtual CLBs must be evaluated in the proper order (see Fig. 2) and the MRs used in a microcyle cannot exceed the number of actual CLBs in a TMFPGA. We call this type of partitioning as *precedence-constrained partitioning*.

By the above constraints, we formulate the MP-constrained partitioning (MPCP) problem for TMFPGAs as follows.

MP-Constrained Partitioning Problem:

Input: Given a circuit $G = (V, E)$ and the number of MPs p in a TMFPGA.

Problem: Determine the p-stage precedence-constrained partitioning with the following objectives.

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

The logic in different stages share the same physical CLBs in a time-multiplexed manner. Hence, the CLBs used in a microcyle cannot exceed the number of actual CLBs in a TMFPGA, i.e., the number of vertices should be smaller than the number of actual CLBs. The balance objective, minimize $\max\{w(V_i) | 1 \le i \le p\}$, 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.

Fig. 3 shows a part of a design that has been partitioned into four MPs in a TMFPGA. Assume that a vertex requires a CLB and an interconnection requires an MR. For example, the partitioning shown in Fig. 3(a) needs five CLBs and five MRs while that shown in Fig. 3(b) uses only three CLBs and three MRs. Therefore, the partitioning shown in Fig. 3(b) is desirable.

III. 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).

A. Two-Terminal Nets

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

In the MPCP problem, given a circuit $G(V, E)$ and a TMFPGA with p MPs, 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	Integer variable that denotes the number of CLBs needed in the TMFPGA.
M_r	Integer variable that denotes the number of MRs (or interconnections) needed in the TMFPGA.
$x_{i,j}$	0–1 integer variable associated with v_i. $x_{i,j} = 1$ if v_i is assigned to the stage j; otherwise, $x_{i,j} = 0$.
$y_{i,k}$	0–1 integer variable associated with a net $e_i = (v_{i_1} \rightarrow \langle v_{i_2} \rangle)$, where $e_i \in E$. $y_{i,k} = 1$ if net $e_i = (v_{i_1} \rightarrow \langle v_{i_2} \rangle)$ introduces an interconnection between the stages k and $k+1$ (kth cut); otherwise, $y_{i,k} = 0$.

Liu and Wong in [15] used an *α-bounded unidirectional 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, $\alpha = |V|/p$.

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:

$$y_{i,k} = (\mathbf{OR}(x_{i_1,1}, \ldots, x_{i_1,k}))\ \mathbf{AND}\ (\mathbf{OR}(x_{i_2,k+1}, \ldots, x_{i_2,p})), 1 \le k \le p-1. \quad (1)$$

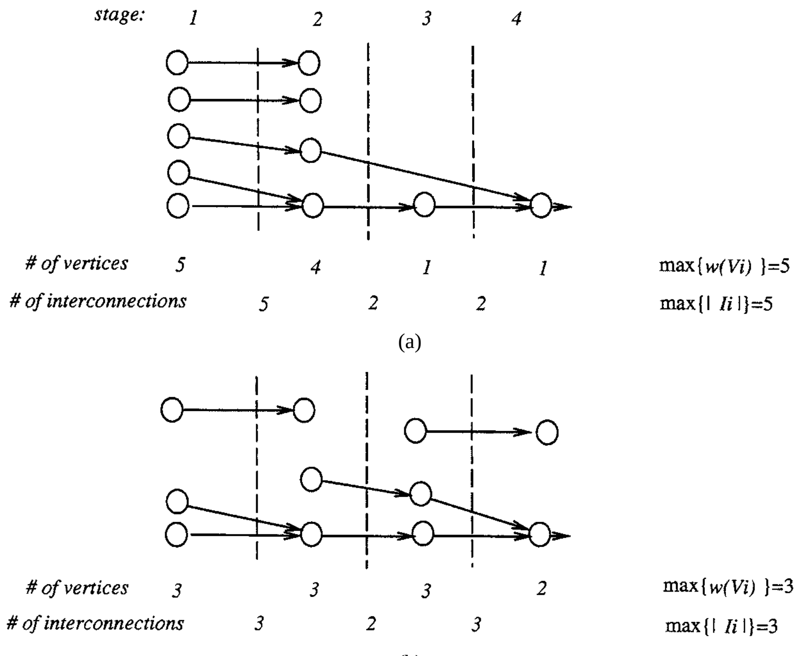


Fig. 3. Precedence-constrained partitioning. (a) $\max\{w(V_i)|1 \le i \le p\} = 5$ and $\max\{|I_i|\,|1 \le i \le p\} = 5$. (b) Better partitioning with $\max\{w(V_i)|1 \le i \le p\} = 3$ and $\max\{|I_i|\,|1 \le i \le p\} = 3$.

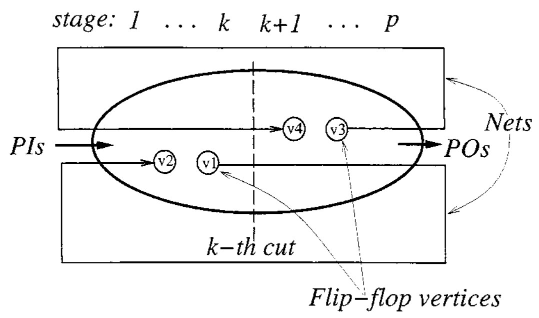


Fig. 4. Two flip-flop nets $(v_1 \to \langle v_2\rangle)$ and $(v_3 \to \langle v_4\rangle)$. Each of them contributes an interconnection to the kth cut.

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 (2)

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

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 $e_i = (v_{i_1} \to \langle v_{i_2}\rangle) \in E_f$, $y_{i,k} = 1$ 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 Fig. 4 for a graphical representation.)

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

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

Similarly, (3) is mathematically equivalent to (4)

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

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

Therefore, the MPCP problem can be formulated as follows:

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

The objective function is used to minimize the number of MRs. Constraint (5) ensures that M_c deviate 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 $e_i = (v_{i_1} \to \langle v_{i_2}\rangle)$, if $e_i \in E_c$ (E_f), then constraint (8) [(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 an MR to save its signal.

B. Multiterminal Nets

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

First, we consider multiterminal combinational nets in a circuit. For a net $e_i = (v_i \rightarrow \langle v_{j_1}, v_{j_2}, \ldots, v_{j_q}\rangle)$ in E_c, v_i is a combinational vertex and $\mathrm{Pre}(v_i) \preceq \mathrm{Pre}(v_{j_t})$ for $1 \le t \le q$. v_i must be in a stage no later than its output vertices $v_{j_t}{}'s$ $(1 \le t \le q)$. Therefore, we shall rewrite constraint (8) for net e_i as follows:

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

For a net $e_i \in E_c$, $y_{i,k} = 1$ if v_i is assigned to stage 1,2,..., 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:

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

Let

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

Rewriting (11), we have

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

Similar to (2), we have

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

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

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

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

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

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

For a net $e_i \in E_f$, $y_{i,k} = 1$ if v_i and any of $v_{j_t}{}'s$ (where $1 \le t \le q$) 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 the equation shown at the bottom of the page.

Let

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

Rewriting the equation at the bottom of the page, we have

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

By (4), we have

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

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

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

Now we consider the pth cut. For any flip-flop net $e_i = (v_i \rightarrow \langle v_{j_1}, v_{j_2}, \ldots, v_{j_n}\rangle)$, $y_{i,p}$ always equals one because the output of v_i must be saved for use in the next user cycle. Thus, we have

$$y_{i,p} = 1, \text{ for each flip-flop nets.}$$

C. Performance-Driven MPCP (PDMPCP) Problem

The critical path in each plane determines the execution time of a TMFPGA. 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, $D = \lceil \mathrm{depth}/p \rceil$, where depth is the critical path of the

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

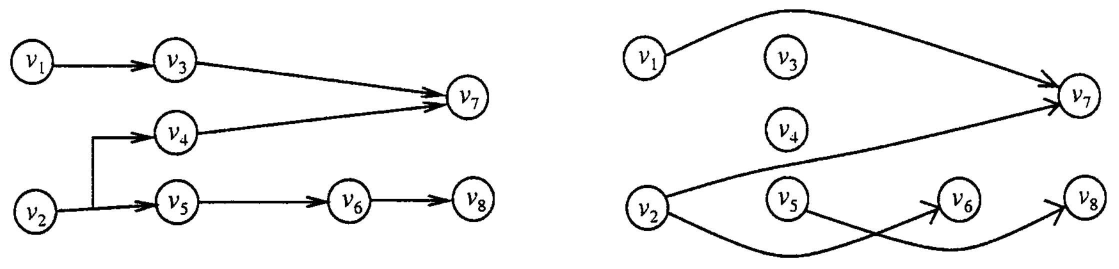


Fig. 5. (a) Piece of combinational part of given circuit. (b) Corresponding TPG with $D = 2$.

circuit. We call the partitioning with a bound of the critical path as *bounded-delay precedence-constrained partitioning*.

The as soon as possible (ASAP) and as late as possible (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:

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

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

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

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 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 $E_t = \{(v_i \to \langle v_j \rangle) |$ pair (v_i, v_j) is critical$\}$. Fig. 5 shows a TPG example that contains eight vertices $v_1, v_2, \ldots, v_8$. Assume that 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 Fig. 5(a), the length of the longest path associated with (v_1, v_7) is equal to three; therefore, they cannot be assigned to the same stage and we connect (v_1, v_7) with a directed edge

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

Subroutine: Timing_Constraint_Edge(path_length, $f$, $t$)
1. if path_length + 1 > $D$ then
2.     $E_t = E_t \cup \{(v_f, v_t)\}$;
3. else
4.     for each immediate successor $v_k$ of $v_t$ in $G$
5.         Timing_Constraint_Edge(path_length + 1, $f$, $k$);
```

Fig. 6. Algorithm for generating TPG.

[see Fig. 5(b)]. Similarly, there are directed edges (v_2, v_7), (v_2, v_6), and (v_5, v_8) in the TPG shown in Fig. 5.

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 $(v_i \to \langle v_j \rangle)$ 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 Fig. 6 generates a TPG associated with $G(V, E)$.

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:

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \le -1$$
$$\text{for each } e_i = (v_{i_1} \to \langle v_{i_2} \rangle),\ e_i \in E_t. \tag{20}$$

Our ILP formulation is summarized in Fig. 7 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 TMFPGA under the precedence and delay-bound constraints.

Problem PDMPCP
Minimize M_r
Subject to

$$(1-\epsilon)\alpha \le M_c \le (1+\epsilon)\alpha \tag{21}$$

$$\sum_{i=1}^{n} x_{i,j} - M_c \le 0, 1 \le j \le p. \tag{22}$$

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

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

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

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

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

$$\sum_{e_i \in E} y_{i,k} - M_r \le 0, 1 \le k \le p. \tag{28}$$

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

Fig. 7. ILP formulation for PDMPCP.

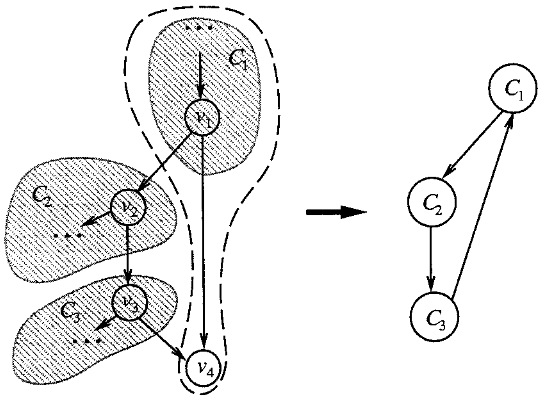
D. Complexity of the PDMPCP Problem

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) in Fig. 7, only n variables can be set to one. Thus, once $x_{i,j}$ is set to one, the remaining variables associated with the range $[A_L(v_i), A_S(v_i)]$ are implicitly set to zero. 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. In Fig. 7, it is obvious that the number of equations required for constraints (21)–(23) and (28) is one, p, n, and p, respectively. For each two-terminal (multiterminal) 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 (multiterminal) 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 $O(n^2 + m')$.

IV. 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 [16] proposed a new clustering metric that is effective in clustering traditional circuits,



(a) (b)

Fig. 8. Clustering that generates a cycle. (a) Example circuit and its clustering. (b) Clustering result.

but may not generate feasible clusters due to the precedence constraints in the TMFPGA partitioning.

Based on Sanker and Rose's algorithm [16], we propose in the following a clustering method that 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 belongs. The score $w_{v,c}$ for each candidate vertex v associated with a cluster c has two components:

1) the number of connections between the candidate vertex v and the cluster c being considered, with each connection weighted by the fan-out of the net on which it lies;
2) 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 fan-out 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 removed.

However, the above procedure might not satisfy the the precedence constraints. It may generate cycles in the graph. For example, in Fig. 8(a), vertices v_1, v_2, and v_3 are clustered in clusters C_1, C_2, and C_3, respectively. Considering the vertex v_4, if the score w_{v_4,c_1} is the highest among the scores associated with v_4, then v_4 will be clustered in C_1, which violates the precedence constraints because this clustering causes a cycle $\langle C_1, C_2, C_3, C_1 \rangle$ as shown in Fig. 8(b). To ensure that no cycle be generated, we cluster according to the topological order of vertices in circuits. Moreover, we check whether there is any cycle created when clustering. The algorithm is named precedence-constrained clustering and is summarized in Fig. 9. 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.

```
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. $V = V - \{v\}$.
```

Fig. 9. Clustering algorithm.

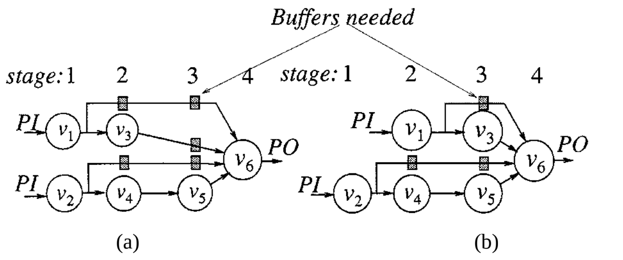


Fig. 10. Buffer registers are needed to store signals among nonadjacent stages. (a) Three buffer registers are needed since $\max\{|B_i| | 1 \le i \le 4\} = 3$ $(|B_3| = 3)$. (b) Only two buffer registers are needed since $\max\{|B_i| | 1 \le i \le 4\} = 2$ $(|B_3| = 2)$.

V. 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 (BRMP) problem that was first investigated in [4] and CLB-constrained stage minimization.

A. Buffer-Register Minimization Partitioning

In this section, we extend our approach to the BRMP problem addressed in [4]. The original problem is addressed on the dharma [1] architecture. As mentioned in [4], buffer registers are needed because the time-multiplexed nature of TMFPGAs 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. Fig. 10 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 Fig. 10(a), while only two buffer registers are required in that shown in Fig. 10(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 fan-out 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 BRMP problem as follows.

Buffer-Register Minimization Partitioning Problem:

Input: Given a circuit $G = (V, E)$ and the number of MPs p in a TMFPGA.

Problem: Determine the p-stage precedence-constrained partitioning with the following objective.

1) 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 multiterminal nets are similar to the techniques presented in Section III-B. Let M_b be a new integer variable that denotes the number of buffer registers needed in the TMFPGA, $z''_{i,k}$ a 0–1 integer variable associated with $e_i = (v_i \rightarrow \langle v_j \rangle)$. $z''_{i,k} = 1$ 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 III-A, we obtain the formulation for the BRMP problem as follows:

minimize $M_c + \alpha M_b$

subject to

$$(1-\epsilon)\alpha \le M_c \le (1+\epsilon)\alpha \tag{30}$$

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

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

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

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

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

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

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

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

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.

B. *CLB-Constrained Stage-Minimization Partitioning*

In the Xilinx TMFPGA, each stage must be stored in a CMP. Each MP is a very large word of memory. Therefore, minimizing the number of stages (i.e., the variable p or the number of MPs needed) allows the design to fit into a TMFPGA with a smaller number of MPs. The significance of minimizing the number of stages required is twofold: 1) it is possible to implement a circuit in a TMFPGA with fewer MPs and 2) a TMFPGA with the fixed number of MPs can accommodate a larger circuit design.

The CLB-constrained stage-minimization partitioning (CCSMP) problem can be described as follows.

CLB-Constrained Stage-Minimization Partitioning Problem:

Input: Given a circuit $G = (V, E)$ and the number of CLBs in a TMFPGA.

Problem: Determine the precedence-constrained partitioning with the following objective.

1) Minimize $p + \alpha M_r$.

The CCSMP problem considers the numbers of stages and interconnections simultaneously. The variables used in the formulations are as follows.

p	Integer *variable* that denotes the number of MPs needed in the TMFPGA. Note that p is a *variable* here while it is a *constant* in the previous discussions.
$y_{i,k}$, M_r, $x_{i,j}$	0–1 integers (the same as the definitions in Section III).

In the CCSMP problem, the maximum number of CLBs is given (fixed). Under this constraint, we can apply the list scheduling to decide the upper bound of the number of stages, p'. Therefore, the CCSMP problem can be formulated as follows:

minimize $p + \alpha M_r$

subject to

$$\sum_{i=1}^{n} x_{i,j} \le M_c, 1 \le j \le p' \tag{39}$$

$$\sum_{j=A_S(v_i)}^{A_L(v_i)} x_{i,j} = 1, 1 \le i \le n \tag{40}$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \le 0 \quad \text{for each } e_i = (v_{i_1} \to \langle v_{i_2} \rangle), \ e_i \in E_c \tag{41}$$

$$\sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} - \sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} \le 0 \quad \text{for each } e_i = (v_{i_1} \to \langle v_{i_2} \rangle), \ e_i \in E_f \tag{42}$$

$$\sum_{k=A_S(v_{i_1})}^{A_L(v_{i_1})} kx_{i_1,k} - \sum_{k=A_S(v_{i_2})}^{A_L(v_{i_2})} kx_{i_2,k} \le -1 \quad \text{for each } e_i = (v_{i_1} \to \langle v_{i_2} \rangle), \ e_i \in E_t \tag{43}$$

$$\sum_{e_i \in E} y_{i,k} - M_r \le 0, 1 \le k \le p' \tag{44}$$

$$\sum_{j=1}^{p'} jx_{i,j} - p \le 0 \quad \text{for each } v_i \text{ without any successor.} \tag{45}$$

TABLE I
MCNC PARTITIONING93 BENCHMARK CIRCUITS

Circuit	#vertices	#Nets	#PIO	Depth
c3540	1038	1016	72	38
c5315	1778	1655	301	30
c6288	2856	2824	64	14
c7552	2247	2140	313	25
s1423	831	750	26	61
s820	340	314	41	12
s838	495	459	41	57
s9234	6098	5846	45	59
s13207	9445	8653	156	60
s15850	11071	10385	105	83
s35932	19880	17830	359	31
s38417	25589	23845	138	84
s38584	22451	20719	284	57

The objective function is used to minimize the numbers of MPs and MRs simultaneously, where α is a user specified parameter. Constraint (39) states that each plane cannot contain more than M_c CLBs. Note that M_c in constraint (39) is a *constant* and no vertex should be assigned to a stage later than p, as described in constraint (45).

VI. Experimental Results

The programs for our system (the ASAP and ALAP scheduling, i.e., bounded-delay precedence-constrained partitioning presented in Section III-C, clustering, and ILPs) were written in the C++ language and the ILPs were solved using the LINDO package [14] 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]–[6] and [15]. Columns 2–4 in Table I list the number of vertices, nets, and primary input–outputs in the circuits, respectively. In column 5, depth refers to the number of vertices on the longest critical path.

We compared our method with the list scheduling list [4], [5], the network-flow-based approach FBP-m [15], and the probability-based approach PAT [6] on the Xilinx TMFPGA model in which a circuit was partitioned into eight stages. The size of a stage is bound by the balance factor 5%. This is the same as in [5], [6], and [15]. The results are shown in Table II. Columns 2–4 in Table II list the maximum numbers of MRs (cuts) used by List, FBP-m, and PAT, respectively. Column 5 lists the maximum numbers of MRs used by our algorithm and the runtimes are shown in brackets. Columns 6–8 list the percentages of improvements of ours over List, FBP-m, and PAT, respectively. The improvement for the List (FBP-m, PAT) is calculated by

$$\frac{\mathrm{List\ (FPB-m,\ PAT) - Ours}}{\mathrm{List\ (FPB-m,\ PAT)}} \times 100\%.$$

The results show that our method on the average reduces the maximum numbers of MRs 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. Our approach is practical and scales well to larger problems. As shown in Fig. 11 in which 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 min for the smallest circuit s820 to about 6 h for the largest circuit s35932. (In the ILP formulation, the numbers of variables and constraints needed by the largest circuit s35932 are 11 328 and

TABLE II
RESULTS FOR THE 8-STAGE TMFPGA PARTITIONING

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

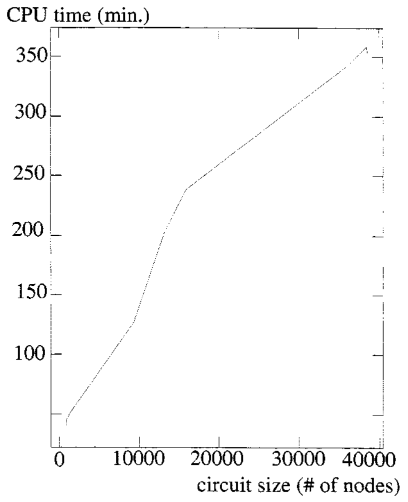


Fig. 11. Runtime requirement versus circuit size.

5046.) 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 h and List (on an Ultra Sparc 1 workstation) consumes about 1.5 h on a different TMFPGA architecture.]

VII. CONCLUSION

We have presented generic ILP formulations for a *set* of multistage 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 TMFPGA partitioning.

ACKNOWLEDGMENT

The authors would like to thank Dr. H. Liu for providing the benchmark circuits and the anonymous reviewers for their constructive comments.

REFERENCES

[1] N. B. Bhat *et al.*, “Performance-Oriented Fully Routable Dynamic Architecture for a Field Programmable Logic Device,” Univ. California, Berkeley, CA, Memo. UCB/RELM93/42, 1993.
[2] F. Brglez, “ACM/SIGDA design automation benchmarks: Catalyst or anathema?,” *IEEE Des. Test*, vol. 10, pp. 87–91, Sept. 1993.
[3] J. Brown *et al.*, “DELTA: Prototype for a First-Generation Dynamically Programmable Gate Array,” MIT, Cambridge, MA, Transit Note 112, 1995.
[4] D. Chang and M. Marek-Sadowska, “Buffer minimization and time-multiplexed I/O on dynamically reconfigurable FPGAs,” in *Proc. Int. Symp. Field Programmable Gate Arrays*, Feb. 1997, pp. 142–148.
[5] D. Chang *et al.*, “Partitioning sequential circuits on dynamically reconfigurable FPGAs,” in *Proc. Int. Symp. Field Programmable Gate Arrays*, Feb. 1998, pp. 161–167.
[6] M. Chao *et al.*, “A clustering- and probability-based approach for time-multiplexed FPGA partitioning,” in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 364–368.
[7] A. DeHon, “DPGA-coupled microprocessors: Commodity ICs for the early 21st century,” in *Proc. IEEE Workshop FPGAs for Custom Computing Machines*, Apr. 1994, pp. 31–39.
[8] J. Cong, Z. Li, and R. Bagrodia, “Acyclic multiway partitioning of boolean networks,” in *Proc. Design Automatation Conf.*, June 1994, pp. 670–675.
[9] S. Dutt and W. Deng, “Partitioning using second-order information and stochastic-gain functions,” in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 112–117.
[10] C. M. Fidducia *et al.*, “A linear-time heuristic for improving network partitions,” in *Proc. Design Automation Conf.*, June 1982, pp. 175–181.
[11] C. Y. Hitchcock III *et al.*, “A method of automatic data path synthesis,” in *Proc. Design Automation Conf.*, June 1983, pp. 484–488.
[12] C. T. Hwang, J. H. Lee, and Y. C. Hsu, “A formal approach to the scheduling problem in high level synthesis,” in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Feb. 1998, pp. 497–504.
[13] D. Jones and D. M. Lewis, “A time-multiplexed FPGA architecture for logic emulation,” in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1995, pp. 495–498.
[14] “LINDO: Linear Interactive and Discrete Optimizer for Linear, Integer and Quadratic Programming Problems,” LINDO Systems, Inc., Chicago, IL, 1999.
[15] H. Liu and D. F. Wong, “Network flow based circuit partitioning for time-multiplexed FPGAs,” in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Feb. 1998, pp. 497–504.
[16] Y. Sankar and J. Rose, “Trading quality for compile time: Ultra-fast placement for FPGAs,” in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 1999, pp. 157–166.
[17] S. Trimberger, “A time-multiplexed FPGA,” in *Proc. IEEE Workshop FPGAs for Custom Computing Machines*, Apr. 1997, pp. 22–28.
[18] Xilinx, Inc., *The Programmable Logic Data Book*. San Jose, CA: Xilinx, 1996.