Temporal Floorplanning Using 3D-subTCG *

Ping-Hung Yuh[1], *Chia-Lin Yang*[1], *Yao-Wen Chang*[2], *Hsin-Lung Chen*[3]

[1]Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
{r91089, yangc}@csie.ntu.edu.tw
[2]Graduate Institute of Electronics Engineering & Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
ywchang@cc.ee.ntu.edu.tw
Etron Technology Inc., Hsin-Chu, Taiwan
gis89536@cis.nctu.edu.tw

Abstract

Improving logic capacity by time-sharing, dynamically reconfigurable FPGAs are employed to handle designs of high complexity and functionality. In this paper, we use a novel topological floorplan representation, named *3D-subTCG* (3-Dimensional sub-Transitive Closure Graph) to deal with the 3-dimensional (temporal) floorplanning/placement problem, arising from dynamically reconfigurable FPGAs. The 3D-subTCG uses three transitive closure graphs to model the temporal and spatial relations between modules. We derive the feasibility conditions for the precedence constraints induced by the execution of the dynamically reconfigurable FPGAs. Because the geometric relationship is transparent to 3D-subTCG and its induced operations, we can easily detect any violation of temporal precedence constraints on 3D-subTCG. We also derive important properties of the 3D-subTCG to reduce the solution space and shorten the running time for 3D (temporal) foorplanning/placement. Experimental results show that our 3D-subTCG based algorithm is very effective and efficient.

1 Introduction

A Field Programmable Gate Arrays (FPGA) is a (re)programmable logic device that implements multilevel logic. Traditionally, an FPGA needs to be reconfigured as a whole. Recently, several vendors have proposed architectures that allow partially dynamic reconfiguration, such as the Xilinx XC4000E FPGAs [18], the Atmel AT6000 Series FPGAs [2], the Xilinx XC6200 Series FPGAs [11], and Xilinx Virtex Series FPGAs [20].

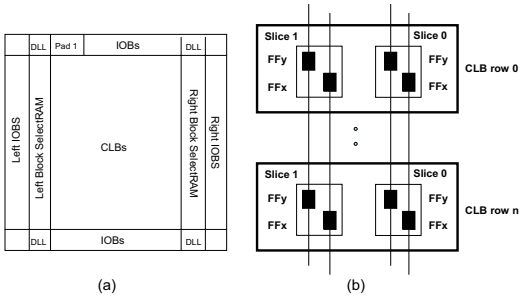

Figure 1: (a) The Virtex architecture. (b) One-column of a 2-slice Virtex CLB.

Figure 1(a) shows the Xilinx Virtex model [20]. The Virtex configuration memory can be considered as an array of bits. The bits of one-bit width that extend from the top to the bottom of the array constitute a vertical *frame*, which is the smallest portion of the configuration memory (i.e., the atomic unit that can be written to or read from in this device). Several frames are grouped together into larger units called *columns*. Figure 1(b) shows one column of *configurable logic blocks* (*CLBs* for short). In such a device, we have to specify a full column of a chip for reconfiguration and read-in/out of flip-flop of contents.

Because of the partial reconfiguration capability in an FPGA, studies have shown that an FPGA-based reconfigurable hardware system can improve performance for many applications [10]. A reconfigurable system is usually composed of a host processor and an FPGA coprocessor, called a *reconfigurable functional unit (RFU)* [3]. An RFU, which can be reconfigured during program execution, may have various configurations at different time. Figure 2(a) shows a program with four parts of codes mapped into RFU operations (called *RFUOPs* or *modules*). Because of the place constraint, we may not load all the modules into the device at the same time. Therefore,

*Yao-Wen Chang's work was partially supported by the National Science Council of Taiwan under Grant No. NSC 91-2215-E-002-038.

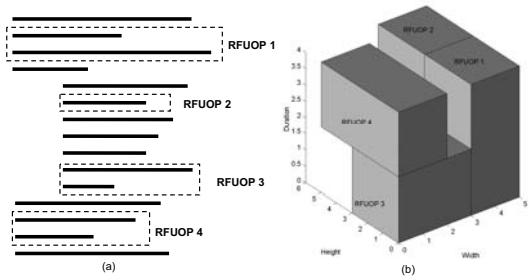

Figure 2: (a) A running program. (b) A 3D-placement of the running program.

how to place these modules into the RFU becomes a 3-D placement problem as shown in Figure 2(b). We may denote each module as a 3-D box with spatial dimensions x and y and the temporal dimension t. There exists temporal relation among scheduled modules since the result of one module may be needed by another one. The objective of temporal floorplanning is to allocate modules in the RFU to optimize the area and execution time without violating the temporal constraints.

1.1 Previous Work

Teich et al. in [17], first used *component graphs* to deal with such a problem assuming no dependence among scheduled modules. They derived necessary and sufficient conditions for a feasible placement and proposed an enumeration scheme by using a branch-and-bound tree search algorithm to find a feasible solution. In practice, however, there often exist temporal precedence constraints among scheduled modules since the output of one module may be needed as the input of another module. Therefore, Fekete et al. in [7] later extended their work to solve the placement problem with temporal precedence constraints by using an additional dependency graph. Bazargan et al. in their pioneering works [3], [4] and [5] considered both offline placement (3D template placement) and online placement. In the offline placement, they modeled each RFUOP as a 3D box and fixed the width and height of the RFU. They proposed a 3D-floorplanner which implements four effective methods, including one greedy method called KAMER-BF (Keep All Maximal Empty Rectangle with Best Fit). In the online placement, they allocated the free space of RFU to an RFUOP dynamically based on different greedy methods (e.g. best-fit and first-fit).

1.2 Our Contribution

In this paper, we solve the 3-dimensional floorplanning/placement problems of the general reconfigurable architecture by using a novel topological floorplan representation, called *3D-subTCG* (3-Dimensional sub-Transitive Closure Graph). To the best knowledge of the authors, this is the first work that uses a topological representation to handle the 3-dimensional placement problem of a dynamically reconfigurable device.

Transitive closure graphs were previously proposed to handle classical 2D floorplanning/placement problems [15]. The main challenge to solve the 3D floorplanning problems is that there exists additional *temporal precedence constraints*, for which some tasks must be executed before other tasks start. We use the 3D-subTCG which consists of three transitive closure graphs to model the temporal as well as the spatial relations between tasks/modules. We derive the feasibility conditions for the temporal precedence and the spatial constraints induced by the execution of the dynamically reconfigurable FPGAs. Because the geometric relationship is transparent to the 3D-subTCG and its induced operations, we can easily detect any violation of temporal precedence and spatial constraints in the 3D-subTCG. Therefore, we can guarantee a feasible solution without resorting

to time-consuming post-processing to remove infeasible ones. We also derive important properties of the 3D-subTCG to reduce the solution space and shorten the running time for 3D (temporal) foorplanning/placement. Experimental results show that our 3D-subTCG based algorithm can obtain significantly better floorplans than the Sequence Triplet (ST) representation (35.18% deadspace in ST v.s. 14.86% in 3D-subTCG). The running-time requirement of 3D-subTCG is also significantly smaller than ST (312.18 sec as ST v.s. 166.46 sec as 3D-subTCG).

The remainder of this paper is organized as follows. Section 2 formulates the temporal floorplanning problem. Section 3 reviews the TCG representation and presents the 3D-subTCG for temporal floorplanning. Section 4 introduces our temporal floorplanning algorithm. Section 5 reports the experimental results. Finally, conclusions are given at Section 6.

2 Formulation

In the reconfigurable architecture, a *task* v is loaded into the device for a period of time for execution. Let $V = \{v_1, v_2, ..., v_m\}$ be a set of m tasks whose widths, heights, and durations are denoted by W_i, H_i, and T_i, $1 \leq i \leq m$. Let (x_i, y_i) $((x'_i, y'_i))$ denote the coordinate of the bottom-left (top-right) corner of a task v_i and , $1 \leq i \leq m$, on the chip. We use t_i (t'_i) to represent the starting (ending) time of v_i, $1 \leq i \leq m$, scheduled in the reconfigurable device.

To guarantee the correctness of the functions in the reconfigurable architecture, we must satisfy temporal precedence requirements, which describe the temporal ordering among tasks. We refer to the temporal precedence requirements as *precedence constraints*. Let $D = \{(v_i, v_j) | 1 \leq i, j \leq m, i \neq j\}$ denote the precedence constraints for the tasks v_i and v_j. The precedence constraints should not be violated during floorplanning/placement.

In order to measure the quality of a floorplan, we consider the following objective functions:

- **Volume (the minimum bounding box of a placement):** In a temporal floorplanning, we need to consider the area of a device and the total execution time tradeoff. If we use a larger device, the total execution time could be shorten. In contrast, it takes longer time if a smaller one is used. Therefore, we shall minimize the product of the area of the device and the total execution time.

- **Wirelength (the summation of half bounding box of interconnections):** Due to the special architecture of the reconfigurable device, the method to estimate the wirelength in the temporal floorplanning is different from the traditional floorplanning/placement problem. Given a net, those nodes in the net may be executed at the same time or at different times. If they are executed at the same time, we can estimate the wirelength according to their geometric distance directly. However, we have to project all nodes into the same time frame before computing their wirelength in the other condition.

- **Communication overhead:** We quantify the communication overhead based on the Xilinx Virtex XCV1000 described in Section 1. Similar to the work by Fekete et al. [7], we assume that a task communicates with another task (data-dependence) in the following way: the results of a CLB, which are read by the successor task, are first written to external memory through a bus interface. The dependent task, which has been loaded at the specified position, then perform a read-in of the results. Recall that a *frame* is the atomic unit that can be written to or read from. Each frame contains 1248 bits and the bus width is only 8 bit. Thus, it takes approximately $1248/8 + 24 = 180$ clock cycles in each read-in or read-out, where the 24 cycles are the configuration overhead of the bus interface as described on the Xilinx FPGA data book [20]. Therefore, the communication overhead of each reconfiguration takes $360 \times f$ clock cycles time (we should first write the data to the external memory and then read back the data) if data in f columns need to be transferred.

- **Reconfiguration overhead:** As described in Section 1, Xilinx Virtex XCV1000 is column-oriented (i.e., all bits in one column should be updated in each read-in or read-out). Suppose that a task v_i occupies $W_i \times H_i$ CLBs. We have to reconfigure H_i columns of CLBs in each reconfiguration. As an example, each CLB column in a Virtex FPGA consists of 48 frames, which takes $(1248/8) \times 48 + 24 = 7512$ clock cycles to configure per CLB column. This means we need $W_i \times 7512$ clock cycles in total if the addresses in the column are incrementally updated.

In this paper, we treat a task v_i as a three-dimensional box. A placement $\mathcal{P}$ is an assignment of (x_i, y_i, t_i) for each v_i, $1 \leq i \leq m$, such that no two boxes overlap and all precedence constraints are satisfied. The goal of temporal floorplanning is to optimize a predefined cost metric (defined in the above)induced by a placement.

3 3D-subTCG for Temporal Floorplanning

3.1 Review of TCG

We first review the TCG representation presented in [15]. TCG uses two graphs, *a horizontal transitive closure graph* C_h and *a vertical transitive closure graph* C_v, to describe the geometric relations among modules. For two non-overlap modules b_i and b_j, b_i is said to be *horizontally* (*vertically*) *related* to b_j, denoted by $b_i \vdash b_j$ ($b_i \perp b_j$), if b_i is on the left (bottom) side of b_j and their projections on *y (x)* axis overlap. For two non-overlap modules b_i and b_j, b_i is said to be *diagonally related* to b_j if b_i is on the left side of b_j, and their projections on the *x* axis and *y* axis do not overlap. To simplify the operations on geometric relations, we treat a diagonal relation as a horizontal one, unless there exists a chain of vertical relations from b_i (b_j), followed by the modules enclosed with the rectangle defined by the two closest corners of b_i and b_j , and finally to b_j (b_i), for which we make $b_i \perp b_j (b_j \perp b_i)$. For each module b_i, we introduce one node n_i both in C_h and C_v. If $b_i \vdash b_j$, a directed edge (n_i, n_j) is constructed in C_h. Similarly, we construct a directed edge (n_i, n_j) in C_v if $b_i \perp b_j$. Figure 3(a) shows a placement with five modules *a*, *b*, *c*, *d*, and *e* whose widths and heights are (2, 1), (2, 2), (3, 2), (1, 2), and (3, 1), respectively. Figure 3(b) shows the TCG corresponding to the placement of Figure 3(a). The weight of each node in C_h (C_v) represents the width (height) of the corresponding module b_i. Since $b_a \vdash b_b$, we construct a directed edge (n_a, n_b) in C_h. Similarly, since $b_a \perp b_c$, a directed edge (n_a, n_c) is constructed in C_v.

TCG has the following three *feasibility properties* [15]:

1. C_h and C_v are acyclic.
2. Each pair of nodes must be connected by exactly one edge either in C_h or in C_v.
3. The transitive closure of C_h (C_v) is equal to C_h (C_v) itself.[1]

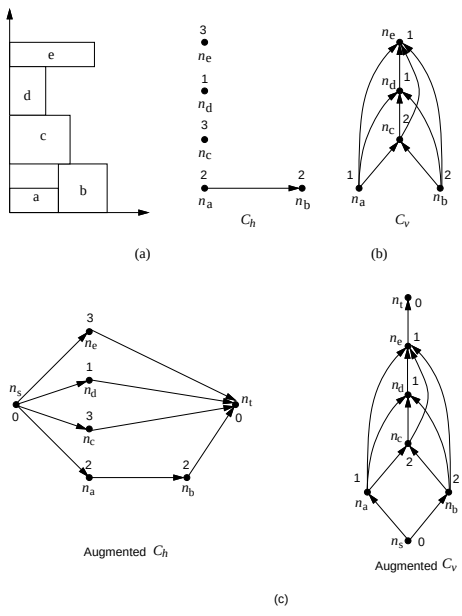

Figure 3: (a) A placement. (b) TCG. (c) Augmented TCG (augmented C_h and C_v).

The first property ensures that a module b_i cannot be both left and right to (below and above) another module b_j in a placement. The second property guarantees that no two modules overlap since each pair of modules have exactly one of the horizontal or vertical relation. The third property eliminates redundant solutions. Figure 4 illustrates the third property. As shown in Figure 4 (a) , since there is a path from node n_c to node n_e in C_v, the edge (n_c, n_e) must be in C_v. If we place the edge (n_c, n_e) into C_h, as shown in Figure 4 (c), the resulting area of placement must be larger or equal to the configuration of Figure 4 (a). Figure 4 (b) and (d) shows the two placement. The third property eliminates this redundant solution.

Given a TCG, a placement can be obtained in $O(m^2)$ time by performing a well-known *longest path algorithm* [14] on TCG, where m is the number of modules. To facilitate the implementation of the longest path algorithm, the two closure graphs can be augmented as follows. For each closure graph, we introduce two special nodes with zero wights, the source n_s and the sink n_t, and construct an edge from n_s to each node with in-degree equal to zero and also from each node with out-degree equal to zero to n_t. Figure 3(c) shows the augmented TCG for the TCG shown in Figure 3(b).

[1] The transitive closure of a directed acyclic graph G is defined as the graph $G' = (V, E')$, where $E' = \{(n_i, n_j)$: there is a path from node n_i to node n_j in $G\}$.

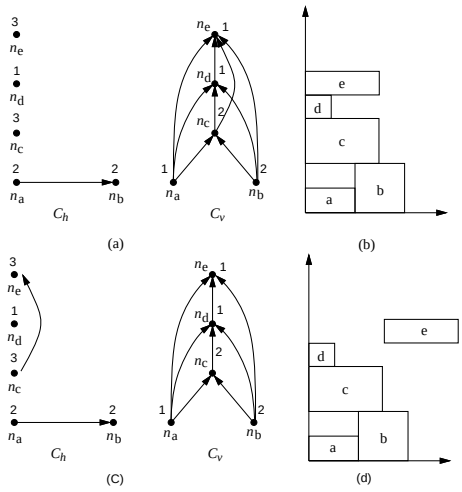

Figure 4: (a) A feasible TCG that the edge (n_c, n_e) lies in C_v. (b) The corresponding placement of Figure 4 (a). (c) A non-feasible TCG that the edge (n_c, n_e) lies in C_h. (d) The corresponding placement of Figure 4 (c).

Let $L_h(n_i)$ ($L_v(n_i)$) denote the weight of the longest path from n_s to n_i in the augmented C_h (C_v). $L_h(n_i)$ ($L_v(n_i)$) can be determined by performing the single source longest path algorithm on the augmented C_h (C_v) in $O(m^2)$ time, where m is number of modules. The coordinate (X_i, Y_i) of a module b_i is given by ($L_h(n_i)$, $L_v(n_i)$). Further, the coordinates of all modules are determined in the topological order in C_h (C_v). Since the respective width and height of the placement for the given TCG are $L_h(n_t)$ and $L_v(n_t)$, the area of the placement is given by $L_h(n_t) \times L_v(n_t)$. Since each module has a unique coordinate after packing, there exists a unique TCG corresponding to any placement.

3.2 3D-subTCG

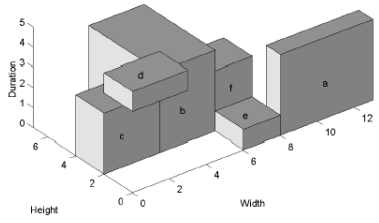

Figure 5: A placement.

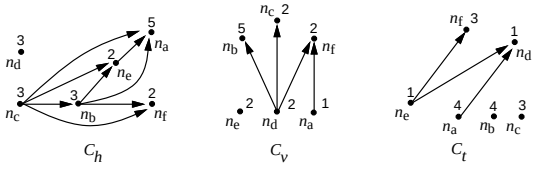

Figure 6: The corresponding 3D-subTCG of Figure 5.

As shown in the previous section, TCG describes the geometric relations among modules based on two graphs, C_h and C_v. For a dynamically reconfigurable device, there exists temporal ordering among tasks. For two tasks v_i and v_j, v_i is said to be *temporally related* to v_j, denoted by $v_i \prec v_j$, if v_i must be executed before v_j starts. To solve the 3D floorplanning/placement problems, we need to consider the temporal and spatial relations at the same time. Therefore, we introduce a new graph to model the temporal relations among tasks, namely a *temporal transitive closure graph* C_t. This new representation is called 3D-subTCG, which contains three transitive graphs, C_h, C_v and C_t. For each task v_i, we construct one node n_i in each graph. If $v_i \vdash v_j$ ($v_i \perp n_j$), we construct one edge (n_i, n_j) in C_h (C_v). If v_i must be executed before v_j, we construct an edge (n_i, n_j) in C_t.

Figure 5 shows a placement with six tasks a, b, c, d, e, and f whose widths, heights and durations are (5, 1, 4), (3, 5, 4), (3, 2, 3), (3, 2, 1), (2, 2, 1), and (2, 2, 3), respectively. Figure 6 shows the 3D-subTCG corresponding to the placement of Figure 5. The value associated with a node in C_h (C_v or C_t) gives the width (height or duration) of the corresponding task, and the edge (n_i, n_j) in C_h (C_v or C_t) denotes the horizontal (vertical or temporal) relation of v_i and v_j. In Figure 6, since task v_c (v_a) is left to (below) v_b (v_f), there exists an edge (n_c, n_b) ((n_a, n_f)) in C_h (C_v). Similarly, since task v_a must be executed before task v_d, there exists an edge (n_a, n_d) in C_t. To obtain the coordinate of each task, we apply the longest path algorithm to the three graphs in a 3D-subTCG. (See Section 3.1 for the details.)

3D-subTCG has the following three *feasibility properties*:

1. C_h, C_v and C_t are acyclic.
2. Each pair of nodes must have exactly one edge either in C_h, C_v or C_t.
3. There must exist an edge (n_i, n_j) if there is a path from n_i to n_j in one graph and there exists no closure edge between n_i and n_j in other graphs.

The first two properties, which are the same as TCG, guarantee that a solution is feasible. The third property is to eliminate the redundant solutions. An edge (n_i, n_j) is said to be a *closure edge* if there exists a path from node n_i to node n_j except the edge (n_i, n_j) itself. For example, the edges (n_b, n_a), (n_c, n_a), (n_c, n_e), and (n_c, n_f) in C_h of Figure 6 are closure edges. If there exists a path from node n_i to node n_j in one graph, the closure edge (n_i, n_j) should appear in the same graph instead of others to eliminate the redundant solutions as explained in section 3.1. However, before adding a new closure edge (n_i, n_j) after each operation, we need to make sure that there exists no closure edges between n_i and n_j in other graphs. Figure 7 illustrates this scenario. Figure 7 (a) shows a 3D-subTCG that node n_a and n_b have a closure edge in C_v. Figure 7 (b) shows the resulting graph after deleting edge (n_e, n_b) in C_t and adding edge (n_b, n_e) to C_h. Now there is a path from n_b to n_a in C_h. However, in order to maintain the second property, we cannot add the closure edge (n_b, n_a) in C_h since (n_a, n_b) has already existed in C_v.

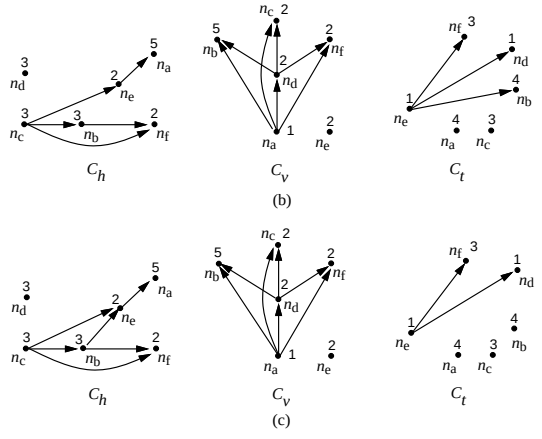

Figure 7: (a) A 3D-subTCG with only one path between node n_a and n_b in C_v. (b) A 3D-subTCG contains two paths in C_h and C_v between node n_a and n_b.

4 Temporal Floorplanning Algorithm

Our algorithm is based on simulated annealing [12]. Given an initial 3D-subTCG, we perturb the 3D-subTCG to obtain a new 3D-subTCG. The cost function Φ used in our algorithm is given by

$$\Phi = \alpha V + \beta W + \gamma O, \tag{1}$$

where V is the volume of the placement, W is the total wirelength, O is the reconfiguration and communication overheads, and α, β, and γ are user-specified constants. In this section, we first describe how to identify a reduction edge, and then show the perturbation operations in simulated annealing. Finally, we introduce the feasibility condition that a 3D-subTCG must satisfy during each perturbation in order to maintain the correct temporal ordering among tasks.

4.1 Reduction Edge Identification

First we illustrate the concept of *reduction edge*. An edge (n_i, n_j) is called *reduction edge* if there does not exist another path from node n_i to node n_j except the edge (n_i, n_j) itself. For example, the edges (n_b, n_f), (n_b, n_e) and (n_e, n_a) in C_h of Figure 6 are reduction edges. Recall that 3D-subTCG is formed by directed acyclic transitive closure graphs. Given an arbitrary node n_i in one transitive closure graph, there exists at least one reduction edge (n_i, n_j), where $n_j \in F_{out}(n_i)$. Here we define the fan-in (fan-out) of a node n_i, denoted by $F_{in}(n_i)(F_{out}(n_i))$, as the nodes n_j's with edges (n_j, n_i) ((n_i, n_j)). For nodes $n_k, n_l \in F_{out}(n_i)$, the edge (n_i, n_k) cannot be a reduction edge if $n_k \in F_{out}(n_l)$. Hence, we remove those nodes in $F_{out}(n_i)$ that are fan-outs of others. The edges between n_i and the remaining nodes in $F_{out}(n_i)$ are reduction edges. In the C_h of Figure 6,

$F_{out}(n_c) = \{n_a, n_b, n_e, n_f\}$. Since n_a, n_e, and n_f belong to $F_{out}(n_b)$, edges (n_c, n_a) and (n_c, n_f) are closure edges while (n_c, n_b) is a reduction one. The reason for identifying reduction edges is that the operations defined below are only applied to reduction edges. The time complexity of finding such a reduction edge is $O(m^2)$, where m is the number of modules (tasks) [15].

4.2 Solution Perturbation

We define the following five operations to perturb a 3D-subTCG:

- *Rotation:* Rotate a task.
- *Swap:* Swap two nodes in C_h, C_v, *and* C_t.
- *Reverse:* Reverse a *reduction edge* in C_h, C_v, *or* C_t.
- *Move:* Move a *reduction edge* from one graph (C_h, C_v, *or* C_t) to another graph.
- *Transpositional Move:* Move a *reduction edge* from one graph (C_h, C_v, *or* C_t) to another graph, and then transpose the two nodes associated with the edge. It is clear later that this operation is different from performing Move followed by Reverse.

Note that Rotation, Swap, Reverse, and Move are first introduced in [15], which can be performed in respective $O(1), O(1), O(m^2)$, and $O(m^2)$ times, where m is the number of modules (tasks). Further, the resulting graph after performing any of these operations on a 3D-subTCG is still a 3D-subTCG. Rotation and Swap do not change the topology of 3D-subTCG, while Revere, Move, and Transpositional Move do. Therefore, to maintain the properties of a 3D-subTCG, we may need to update the resulting graphs after performing Reverse, Move and Transpositional Move. Further, in order to guarantee that the precedence constraints are not violated by these operations, we shall perform feasibility detection, which are described in section 4.3. We first detail the operations in the following.

4.2.1 Rotation

To rotate a *task* v_i, we only need to exchange the weights of the corresponding nodes n_i in C_h, C_v, and C_t. Figure 8 (b) shows the result after rotating the module a in Figure 8.

4.2.2 Move

The Move operation moves a *reduction* edge (n_i, n_j) in one graph to one of the others in a 3D-subTCG. Move could switch the relations of the two tasks v_i and v_j between a horizontal relation and a vertical one. For two tasks v_i and v_j, $v_i \vdash v_j$ ($v_i \perp v_j$) if there exists a reduction edge (n_i, n_j) in C_h (C_v); after moving the edge (n_i, n_j) to C_v (C_h), we have the new geometric relation $v_i \perp v_j$ ($v_i \vdash v_j$). Move could also change the temporal relation of the two tasks v_i and v_j. For two tasks v_i and v_j, $v_i \prec v_j$ if there exists a reduction edge (n_i, n_j) in C_t; after moving the edge (n_i, n_j) to C_h (C_v), we change the temporal relation into the new geometric relation $v_i \vdash v_j$ ($v_i \perp v_j$). If there exists a reduction edge (n_i, n_j) in C_h (C_v); after moving the edge (n_i, n_j) to C_t, we have the new temporal relation $v_i \prec v_j$.

To move a reduction edge (n_i, n_j) from one graph G to another graph G', we first delete the edge (n_i, n_j) from G and then add (n_i, n_j) to G'. For each node $n_k \in F_{in}(n_i) \cup \{n_i\}$ and $n_l \in F_{out}(n_j) \cup \{n_j\}$, we shall check whether the edge (n_k, n_l) exists in G'. If G' contains the edge, we do nothing; otherwise, we need to add the edge to G' and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in G or G'', if any, to maintain the properties of the 3D-subTCG. Figure 8 (c) shows the result of moving the edge (n_c, n_b) in C_h of Figure 8 (b) to C_t.

4.2.3 Swap

To swap nodes n_i and n_j of two tasks v_i and v_j, we only need to exchange the nodes n_i and n_j in C_h, C_v, and C_t. Figure 9 (a) shows the result of swapping nodes n_b and n_d shown in Figure 8 (c).

4.2.4 Reverse

The Reverse operation reverses the direction of a *reduction* edge (n_i, n_j) in one graph. For two modules v_i and v_j, $v_i \vdash v_j$ ($v_i \perp v_j$) if there exists a reduction edge (n_i, n_j) in C_h (C_v); after reversing the edge (n_i, n_j), we have the new geometric relation $v_j \vdash v_i$ ($v_j \perp v_i$). Similarly, $v_i \prec v_j$ if there exists a reduction edge (n_i, n_j) in C_t; after reversing the edge (n_i, n_j), we have the new temporal relation $v_j \prec v_i$.

To reverse a reduction edge (n_i, n_j) in a graph, we first delete the edge from the graph, and then add the edge (n_j, n_i) to the same graph. Similar to the Move operation, for each node $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{out}(n_i) \cup \{n_i\}$ in the new graph, we shall check whether the edge (n_k, n_l) exists in the new graph. If the graph contains the edge, we do nothing; otherwise, we need to add the edge to the graph and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in the other transitive closure graphs, if any, to maintain the properties of the 3D-subTCG. Figure 9 (b) shows the result after reversing the edge (n_e, n_a) in C_h of Figure 9 (a).

4.2.5 Transpositional Move

The Transpositional Move operation removes a *reduction* edge (n_i, n_j) from one graph, and add an edge (n_j, n_i) to one of the others in a 3D-subTCG.

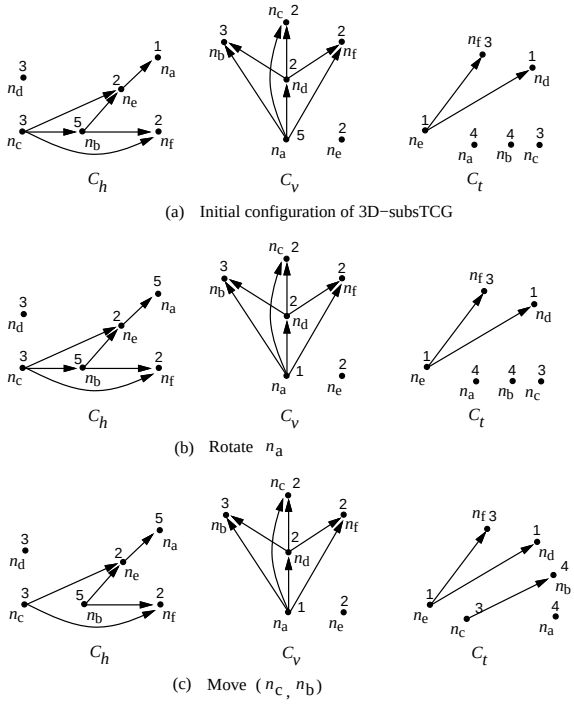

Figure 8: Examples of perturbations. (a) The initial 3D-subTCG (C_h, C_v, and C_t). (b) The resulting 3D-subTCG after rotating the task n_a shown in(a). (c) The resulting 3D-subTCG after moving the reduction edge (n_c, n_b) from the C_h of (b) to C_t.

In one case, Transpositional Move switches the geometric relation of the two tasks v_i and v_j between a horizontal relation and a vertical one and changes the ordering of the two tasks v_i and v_j in their geometric relation. For two tasks v_i and v_j, $v_i \vdash v_j$ ($v_i \perp v_j$) if there exists a reduction edge (n_i, n_j) in C_h (C_v); after transpositionally moving the edge (n_i, n_j) to C_v (C_h), we have the new geometric relation $v_j \perp v_i$ ($v_j \vdash v_i$). In the other case, Transpositional Move changes the temporal relation of the two tasks v_i and v_j. For two tasks v_i and v_j, $v_i \prec v_j$ if there exists a reduction edge (n_i, n_j) in C_t; after transpositionally moving the edge (n_i, n_j) to C_h (C_v), we change the temporal relation into the new geometric relation $v_j \vdash v_i$ ($v_j \perp v_i$). If there exists a reduction edge (n_i, n_j) in C_h (C_v); after transpositionally moving the edge (n_i, n_j) to C_t, we have the new temporal relation $v_j \prec v_i$.

To transpositionally move a reduction edge (n_i, n_j) from one graph G to another graph G', we first delete the edge (n_i, n_j) from G and add (n_j, n_i) to G'. Similar to the Move operation, for each node $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{out}(n_i) \cup \{n_i\}$, we shall check whether the edge (n_k, n_l) exists in G'. If G' contains the edge, we do nothing; otherwise, we need to add the edge to G' and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in G or G'', if any, to maintain the properties of the 3D-subTCG. Figure 9 (c) shows the result of transpositionally moving the edge (n_e, n_b) from C_t of Figure 9 (b) to C_v. Note we delete the edge (n_a, n_e) in C_h and add it to C_v.

4.3 Feasibility Detection

To maintain the temporal ordering among tasks, the 3D-subTCG must guarantee that all precedence constraints are satisfied. Among the five operations mentioned above, Move, Swap, Reverse, and Transpositional Move could violate the constraints. We now show how to detect a violation during perturbation.

When we move an edge (n_i, n_j) or reverse/transpositionally move (n_j, n_i), the precedence constraint will be violated if $n_l \in F_{in}(n_i) \cup \{n_i\}$, $n_k \in F_{out}(n_j) \cup \{n_j\}$, and $(n_l, n_k) \notin C_t$ since $(n_l, n_k) \in D$. As mentioned in Section 2, D denotes the precedence constraints. When we swap two nodes n_i and n_j, three scenarios could happen:

1. there exists a precedence constraint between n_i and n_j,
2. neither of n_i and n_j has a precedence constraint, or
3. either n_i or n_j has precedence constraint.

In the first case, it is clear that we cannot swap the two nodes. However, if neither of n_i and n_j has a precedence constraint, we can swap n_i and n_j directly. Without loss of generality, we could assume that node n_i has precedence constraints to detail the third case. If node n_j has precedence constraints, we can apply the same approach to check the feasibility. In the first condition, n_i has a precedence-constrained edge (n_i, n_k), we can swap n_i and n_j without any violation if $n_k \in F_{out}(n_j)$ in C_t. On the other hand, if n_i has a precedence-constrained edge (n_k, n_i) and $n_k \in F_{in}(n_j)$ in C_t, we can also swap n_i and n_j.

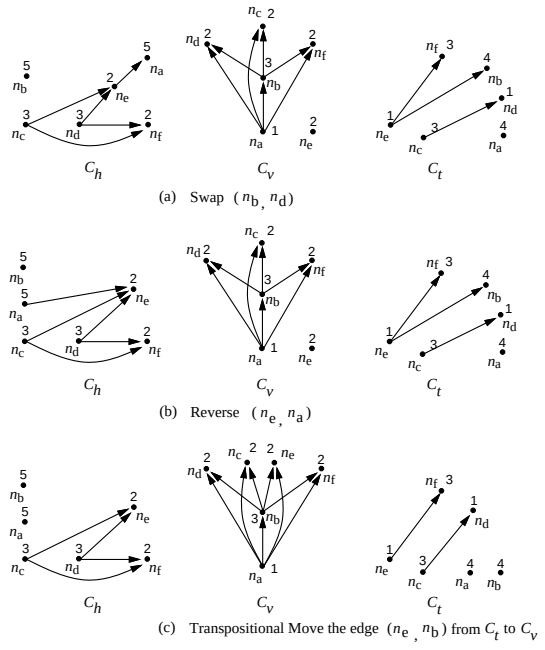

Figure 9: Examples of perturbations (continued from Figure 8). (a) The resulting 3D-subTCG after swapping the nodes n_b and n_d shown in Figure 8(c). (b) The resulting 3D-subTCG after reversing the reduction edge (n_e, n_a) in the C_h shown in (a). (c) The resulting 3D-subTCG after transpositional moving the reduction edge (n_e, n_b) from the C_t of (b) to C_v.

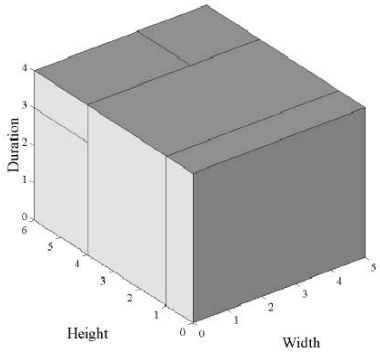

Figure 10: The resulting placement of 3D-subTCG in Figure 9(c).

Figure 11(a) shows the resulting C_h, C_v, and C_t after swapping the nodes n_d and n_e in Figure 9(c). Assume that there exists a precedence-constrained edge (n_e, n_f). The precedence constraint will be violated if we swap the two nodes n_d and n_e since $n_f \notin F_{out}(n_d)$ in the C_t. Figure 11(b) shows C_h, C_v, and C_t after reversing the edge (n_d, n_e) in the C_h in Figure 9(c). Since $\{n_e\} \cap F_{in}(n_e)$=$\{n_c, n_e\}$ and $\{n_d\} \cap F_{out}(n_d)$=$\{n_d, n_f\}$ in C_h, we shall check (n_e, n_f) for the precedence constraint. If there exists a precedence-constrained edge (n_e, n_f), the precedence constraint will be violated.

By doing the feasibility detection during the Move, Reverse, Transpositional Move, or Swap operations, we can guarantee that the resulting 3D-subTCG still satisfies all the precedence constraints. We thus have the following theorem.

Theorem 1 *The precedence constraints of a 3D-TCG are not violated by the Move, Swap, Reverse, or Transpositional Move operation with the feasibility detection.*

5 Experimental Results

Circuit	# of tasks	Sum of volume	Volume	Dead space (%)	time (Sec.)
Circuit 1	10	512	512	0.0	8.3
Circuit 2	10	480	480	0.0	1.9
Circuit 3	10	1000	1000	0.0	9.7
Circuit 4	20	3840	4032	4.7	25.2
Circuit 5	30	4096	4608	11.1	127.8

Table 1: Results of volume optimation (volume = mm^2 x clock cycles).

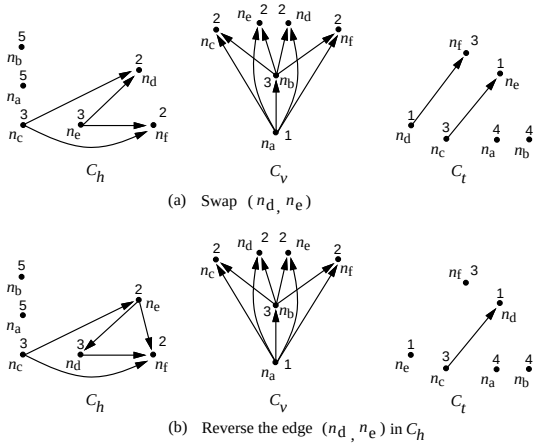

Figure 11: (a) The resulting 3D-subTCG after swapping the nodes n_d and n_e shown in Figure 9(c). (b) The resulting 3D-subTCG after reversing the reduction edge (n_d, n_e) in the C_h shown in Figure 9(c).

Based on simulated annealing [12], we implemented the temporal floorplanning algorithm in the C++ programming language on a 433 MHz SUN Ultra-60 workstation with 1 GB memory. We compared 3D-subTCG with Sequence Triplet (ST). ST is extended from the well-known Sequence Pair (SP) [16], which is very popular for handling floorplanning/placement in both industry and academia [2] A sequence triplet consists of three module sequences (Γ_x, Γ_y, Γ_z). The relation between two modules is defined as follows: (1) if the sequence of two modules a, b is the same (from left to right) in (Γ_x, Γ_z), i.e., (Γ_x, Γ_y, Γ_z) = (..a..b.., ..., ..a..b..), it means that module a is on the Z^+ direction of module b; (2) if the sequence of the two modules a, b is not the same in (Γ_x, Γ_z), the (Γ_x, Γ_y, Γ_z) is identical to Sequence Pair (Γ_x, Γ_y). For example, the ST representation of Figure 5 is $(dcbfea, cbeafd, dafebc)$. Based on the same simulated annealing scheme as that for 3D-subTCG, ST employs the following three perturbation operations: (1) M1: randomly swap two modules in one of the Γ_X, Γ_y, and Γ_z sequences; (2) M2: randomly swap two modules in Γ_X, Γ_y, and Γ_z simultaneously; (3) M3: randomly choose one module and change its height with width, width with length, or length with height (i.e., 3D rotation). We implemented the ST algorithm with the same simulated annealing engine as that of 3D-subTCG with the limiting that rotation can only change width and height (i.e., duration remains the same), and added precedence constraints, reconfiguration overheads and communication overheads for comparative studies.

To verify our algorithm, we first tested 3D-subTCG on five synthetic circuits that can be packed without deadspace. Table 1 shows the results. Note that the volume of a placement is the minimum bounding box enclosing the placement. We can see that 3D-subTCG obtains the optimal placements for the first three test cases and near optimal solutions for the last two larger circuits, all in reasonable time. The results show that our approach is very effective for cost optimization.

Circuit	# of modules	# of pads	# of nets	# pins	# of precedence constraints
3D-apte	9	73	97	214	3
3D-xerox	10	107	203	696	3
3D-hp	11	43	83	264	3
3D-ami33	33	42	123	480	7
3D-ami49	49	24	408	931	11

Table 3: The five 3D-MCNC benchmark circuits.

To compare 3D-subTCG with ST, we performed two experiments. In each experiment, we set $\alpha = \beta\gamma = 1$. In the first experiment, our objective is to minimize the volume with reconfiguration and communication overheads. For this experiment, we adopted the benchmark circuits used in [8] and added the reconfiguration and communication overheads. As shown in Table 2, the 3D-subTCG based method outperforms the ST-based one by a large margin. For example, 3D-subTCG achieved an average deadspace of only 19.38% while ST resulted in an average deadspace 32.01%.

The second experiment is intended to test the 3D placement with the considerations of precedence constraints, wirelength, and reconfiguration/communication overheads. For this experiment, we used the MCNC benchmarks. Since the MCNC benchmarks do not have execution times and precedence constraints, we assigned their execution times and precedence constraints by ourselves. The new benchmark suite is called the 3D-MCNC

[2]The work [16] has been selected as one of the 40 best papers published at ICCAD during the past 20 years [13].

Circuit	(# of tasks)	Sum of Volume	ST			3D-subTCG		
			Volume (mm^2 x clockcycles)	Dead Space (%)	Time (sec.)	Volume (mm^2 x colckcycles)	Dead Space (%)	Time (sec.)
beasley1	10	6218	8710	28.6	7.7	7504	17.1	8.5
beasley2	17	11497	14664	21.5	45.2	12402	7.2	28.5
beasley3	21	10362	16016	35.3	44.1	12640	18.0	22.4
beasley4	7	10205	13800	26.0	3.0	13064	21.8	2.0
beasley5	14	16734	22750	26.4	18.2	18912	11.5	16.0
beasley6	15	11040	14994	26.3	27.9	13200	16.3	24.8
beasley7	8	17168	24570	30.1	3.8	20574	16.5	2.3
beasley8	13	83044	132275	37.2	15.4	98280	15.5	19.4
beasley9	18	133204	174496	23.6	30.6	167751	20.5	17.2
beasley10	13	493746	660480	25.2	13.0	575685	14.2	10.8
beasley11	15	383391	486381	24.8	17.5	438702	12.6	9.8
beasley12	22	646158	922080	29.9	100.0	823816	21.5	58.5
okp1	50	1.24×10^8	2.16×10^8	42.6	1607.2	1.73×10^8	28.4	387.3
okp2	30	8.54×10^7	1.28×10^8	33.2	285.3	1.10×10^8	22.3	73.8
okp3	30	1.23×10^8	1.85×10^8	33.1	280.7	1.60×10^8	23.0	70.6
okp4	61	2.38×10^8	4.17×10^8	42.8	791.3	3.28×10^8	27.3	501.9
okp5	97	1.89×10^8	4.48×10^8	57.7	607.8	2.95×10^8	35.8	565.9
Average				32.01			19.38	

Table 2: Results for volume optimization with reconfiguration overhead and communication overhead.

Circuit	Total volume	ST				3D-subTCG			
		Volume (mm^2 x clockcycles)	Wirelength (mm)	Dead space (%)	Time (sec.)	Volume (mm^2 x clockcycles)	Wirelength (mm)	Dead space (%)	time (sec.)
3D-apte	9.88×10^7	1.18×10^8	495.0	16.2	7.7	1.05×10^8	335,3	5.9	3.9
3D-xerox	4.05×10^7	5.27×10^7	613.2	23.1	19.5	4.42×10^7	602.0	8.4	8.9
3D-hp	1.29×10^7	2.06×10^7	387.3	37.2	20.6	1.50×10^7	158.3	13.7	11.2
3D-ami33	2.32×10^6	4.18×10^6	84.7	44.5	446.4	3.08×10^6	77.7	24.7	128.1
3D-ami49	1.32×10^8	2.93×10^8	1040.8	54.9	1066.7	1.68×10^8	807.1	21.6	680.2
Average				35.18	312.18			14.86	166.46

Table 4: Results of volume and wirelength optimization for the five 3D-MCNC benchmark circuits.

benchmark. Table 3 lists the statistics of the five 3D-MCNC benchmarks. For this experiment, we simultaneously optimized volume and wirelength with precedence constraints, and reconfiguration/communication overheads. Table 4 shows the results. As shown in Table 4, 3D-subTCG achieved better volume utilization (15% deadspace v.s. 35% deadspace) and shorter wirelength compared to ST. 3D-subTCG also needed less CPU time than ST. Figure 12 shows the resulting placement of 3D-xerox.

Although it is hard to quantify, a key insight to the different performance between 3D-subTCG and Sequence Triplet (ST) lies in the effects of their perturbations: swapping two modules in an ST may lead to a dramatic change from the original placement while the change for the 3D-subTCG perturbation is smaller, which makes simulated annealing easier to converge to an optimal solution. (Here is an analogy: Like the gradient search for the optimization of nonlinear programming, the step size plays an important role in determining whether a search scheme can converge to the global optimal solution—a huge step size may fail to converge to an optimal solution.)

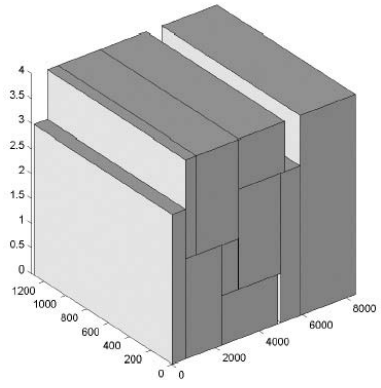

Figure 12: The result of 3D-xerox with optimizing volume and wirelength simultaneous.

6 Conclusion

We have presented the 3D-subTCG representation to handle the temporal floorplanning/placement problem for dynamically reconfigurable FPGAs. We have explored the feasibility conditions for the temporal relations among tasks/modules. Our algorithm can guarantee a feasible placement in each perturbation. Experimental results have shown that our method is very effective and efficient for temporal floorplanning/placement.

Acknowledgements

This work is supported in part by the National Science Council under Grand NSC 92-2213-E-002-014- and NSC 92-215-E-002-043-. We would also thank to anonymous reviewers.

References

[1] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, 1990.

[2] Atmel, "AT6000 FPGA Configuration Guide," Atmel, Inc.

[3] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design & Test of Computers*, vol.17, no. 1, pp. 68–83, Mar. 2000.

[4] K. Bazargan and M. Sarrafzadeh, "Fast Online Placement for Reconfigurable Computing Systems," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 300-302, 1999.

[5] K. Bazargan, R. Kastner and M. Sarrafzadeh, "3-D Floorplanning: Simulated Annealing and Greedy Placement Methods for Reconfigurable Computing Systems," *Design Automation for Embedded Systems - RSP'99 Special Issue*, Apr. 2000.

[6] J. E. Beasley, "An Exact Two-Dimensional Non-Guillotine Cutting Stock Tree Search Procedure," *Operations Research*, vol.33, no. 1, pp. 49–64, 1985.

[7] S. P. Fekete, E. Köhler, and J. Teich, "Optimal FPGA Module Placement with Temporal Precedence Constraints," *Proc. DATE*, pp. 658-665, Mar. 2001.

[8] S. P. Fekete, and J. Schepers, "On more-dimensional packing III: Exact Algorithms," *ZPR Technical Report 97-290* 1997.

[9] M. Gokhale, B. Holmes, A. Kopster, D. Kunze, D. Lopresti, S. Lucas, R. Minnich, and P. Olsen, "Splash: A Reconfigurable Linear Logic Array," *International Conference on Parallel Processing*, pp. 526–532, 1990.

[10] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems," *Proc. of the IEEE*, vol.86, no. 4, pp. 615–639, Apr. 1998.

[11] S. Hauck, Z. Li, and E.J. Schwabe, "Configuration Compression for the Xilinx XC6200 FPGA," *Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 138–146, 1998.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp.671–680, May, 1983.

[13] A. Kuehlmann, Ed., *The Best of ICCAD—20 Years of Excellence in Computer-Aided Design*, Kluwer Academic Pub., 2003.

[14] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976.

[15] J.-M. Lin and Y.-W Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans," *Proc. DAC*, pp. 764–769, June 2001.

[16] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing Based Module Placement," *Proc. ICCAD*, pp. 472–479, 1995.

[17] J. Teich, S. P. Fekete, and J. Schepers, "Compile-Time Optimization of Dynamic Hardware Reconfigurations," *Proc PDPTA*, pp. 1097–1103, June 1999.

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

[19] Xilinx, "XC6200 Field Programmable Gate Arrays Data Sheet," Xilinx, Inc., Oct. 1996.

[20] Xilinx, "XAPP151 Virtex Series Configuration Architecture User Guide v1.5," Xilinx, Inc., Sep. 2000.