VLSI DESIGN
1996, Vol. 4, No. 4, pp. 309–328
Reprints available directly from the publisher
Photocopying permitted by license only

A Timing-Driven Partitioning System for Multiple FPGAs

KALAPI ROY and CARL SECHEN

Department of Electrical Engineering, University of Washington, Seattle, WA 98195

Field-programmable systems with multiple FPGAs on a PCB or an MCM are being used by system designers when a single FPGA is not sufficient. We address the problem of partitioning a large technology mapped FPGA circuit onto multiple FPGA devices of a specific target technology. The physical characteristics of the multiple FPGA system (MFS) pose additional constraints to the circuit partitioning algorithms: the capacity of each FPGA, the timing constraints, the number of I/Os per FPGA, and the pre-designed interconnection patterns of each FPGA and the package. Existing partitioning techniques which minimize just the cut sizes of partitions fail to satisfy the above challenges. We therefore present a timing driven *N*-way partitioning algorithm based on simulated annealing for technology-mapped FPGA circuits. The signal path delays are estimated during partitioning using a timing model specific to a multiple FPGA architecture. The model combines all possible delay factors in a system with multiple FPGA chips of a target technology. Furthermore, we have incorporated a new dynamic net-weighting scheme to minimize the number of pin-outs for each chip. Finally, we have developed a graph-based global router for pin assignment which can handle the pre-routed connections of our MFS structure. In order to reduce the time spent in the simulated annealing phase of the partitioner, clusters of circuit components are identified by a new linear-time bottom-up clustering algorithm. The annealing-based *N*-way partitioner executes four times faster using the clusters as opposed to a flat netlist with improved partitioning results. For several industrial circuits, our approach outperforms the recursive min-cut bi-partitioning algorithm by 35% in terms of nets cut. Our approach also outperforms an industrial FPGA partitioner by 73% on average in terms of unroutable nets. Using the performance optimization capabilities in our approach we have successfully partitioned the MCNC benchmarks satisfying the critical path constraints and achieving a significant reduction in the longest path delay. An average reduction of 17% in the longest path delay was achieved at the cost of 5% in total wire length.

Keywords: FPGA, partitioning, pin assignment, clustering

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are becoming a mainstream technology in board, system and application specific integrated circuit (ASIC) design processes. However, design complexity will continue to increase more rapidly than the availability of larger and faster devices. System-level ASIC designers are turning to FPGAs for design verification to take advantage of their low cost and fast prototyping. Large and complex designs can require multiple iterations in order to achieve a successful design imple-

mentation. If the automatic design tools cannot provide a feasible solution, the designer is forced to obtain expert level architectural knowledge to support the manual intervention required to complete the design. Current FPGA architectures can handle a maximum of only 6000 to 9000 gates compared to ASIC devices which offer hundreds of thousands. Designers utilize multiple FPGAs when a single FPGA is not sufficient for a design implementation.

Multiple re-programmable FPGAs have been configured on multichip modules (Figure 1) and on boards (Figure 2). A large design can be implemented on a PCB [1] using multiple chips of a particular target FPGA architecture such as Xilinx [2], Actel, or Altera. New field programmable architectures using multiple FPGAs on multichip modules have emerged to offer utilization requirements for prototyping large designs. A *multiple FPGA system* (MFS) can be modeled as a collection of FPGA chips configured on a single board or a package to realize a design.

In order to effectively use MFSs and benefit from shorter time-to-market, users require an automatic method to partition a large design among multiple FPGAs. This process could be viewed as a divide-and-conquer process in order to speed the placement and routing phases or as a conventional top-down design process. Each chip in this N-(multiple) chip combination is considered a *partition*.

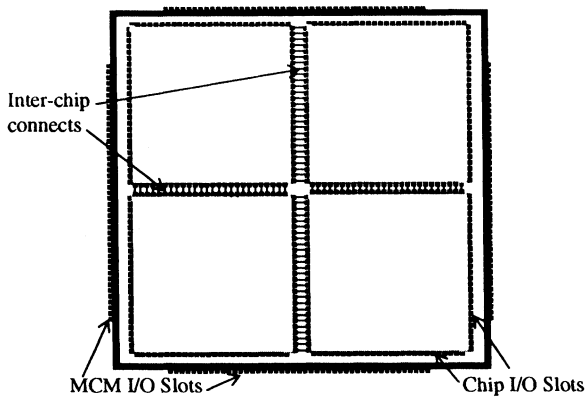

FIGURE 1 An MFS on an MCM.

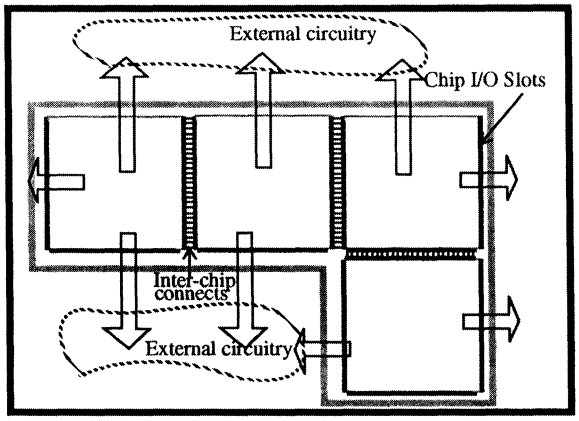

FIGURE 2 An MFS on a PCB.

1.1. Constraints and Objectives of the MFS Partitioning System

Any decision made early in the design process will affect the performance of the subsequent design tools. The quality of the partitioning results will influence several aspects of the design implementation:

1) *Capacity*: The target FPGA architecture used in the MFS has a maximum gate capacity. However, the amount of logic in each chip is limited by the utilization levels that can be handled by the placement and routing tools. Thus, the *feasible* utilization levels are always some fraction of the maximum gate capacity. The partitioner must ensure that each chip contains a feasibly implementable amount of logic.
2) *Congestion in inter-chip communication*: The partitioner must be able to minimize the amount of inter-chip communications. The signals external to individual chips must be routed using the limited number of inter-chip connections to produce a feasible partitioning solution. The fixed inter-chip connections of the packaging or the board design restrict the flexibility of routing any signals which are external to a chip. This process can be viewed as *pin assignment* at the chip level. Any overflow generated during pin assignment will lead to a design which is not implementable.

3) *Delay introduced for intra-chip and inter-chip communications*: High utilization of logic in individual chips causes congestion in intra-chip routing. This leads to longer paths and thus longer delays for signals internal to the chips. Also, the signals which cross one or more chip boundaries in the MFS will accumulate a substantial amount of delay associated with the I/O buffers and the inter-chip wire. Depending on the application, this delay can range from high, as in the case of PCBs, to moderate as in MCM based systems. The system cycle time will be determined by the length of the longest path from a primary input to the primary output of the entire MFS. The partitioner must satisfy the timing specifications for the MFS.

Thus the constraints of the MFS partitioning problem are: 1) Set of FPGA chips with their locations, dimensions and maximum capacity; 2) Configurations of the chip level I/O frames for inter-chip signals; 3) Configurations of the MFS package level I/O slots for the system I/O signals. In addition, the following design constraints need to be satisfied during MFS partitioning: 4) The timing constraints of the system being implemented; 5) Additional user constraints such as the utilization levels within the chips and the preplaced logic which must remain within a particular chip.

The ASIC and board designs which will typically be implemented in an MFS are large and complex. Therefore, it's important to focus on the speed of the partitioning system. A fast partitioning will be compatible with the rest of the design flow of rapid prototyping, one of the major advantages of the MFS.

MFS partitioning over multiple chips can be performed before technology mapping onto the target FPGA or after technology mapping. If the partitioner manipulates the gate level netlist (as in [3]) before technology mapping, estimation of chip utilization and routability is difficult without the exact count of the target technology logic blocks. These estimations are therefore made conservatively to ensure successful execution of place and route tools. At this level, there is no information regarding delay of components and interconnects. Thus, the partitioner will not be able to exercise any timing driven capabilities for an unmapped circuit. This is a major limitation because timing problems and achieving minimum system delay is very important for large complex designs which would typically be partitioned over multiple FPGAs.

After technology mapping, the partitioner can take into account the target FPGA technology specific details such as total logic block count, the routing resources associated with each group of logic blocks assigned to each partition and the actual timing information during the partitioning process. Hence, the partitioning process should follow the technology mapping stage.

2. PREVIOUS WORK

The previous work in partitioning can be classified into three categories. *Netlist partitioning* includes classical partitioning of a netlist into sub-netlists with the goal of minimizing the communication between the sub-netlists. No physical assignment of these sub-netlists is involved during partitioning i.e. in the floor-planning senses. Several improved versions of the classical mincut algorithm by Kernighan-Lin [4] with enhancements by Fiduccia and Mattheyses [5] have been reported [6,7,8,9,10]. *Rectilinear partitioning* is sometimes viewed as floorplanning. It consists of physical partitioning of the rectilinear regions on a chip in conjunction with netlist partitioning to partition the circuit for its placement into these regions. Hierarchical placement techniques [14,15] for physical design are based on rectilinear partitioning. *Multichip partitioning* is a relatively new area of research. This problem is the basic rectilinear partitioning problem with additional constraints involved in a multichip system. Timing-driven system partitioning approaches were reported [16] based on a TCM (Thermal Conduction Module) MCM technology. These approaches were based on extended N-way Kernighan-Lin partitioning with a cost function including both capacity and timing constraints. The problem of pin assignment was not handled by this

approach. The Anyboard rapid prototyping system [2] was created at NCSU for the development and rapid implementation of digital hardware designs. This system consists of the Anyboard PC card which consists of multiple Xilinx FPGAs.

Several commercial vendors have developed tools for partitioning FPGAs. The Prism software tool [17] from NeoCAD provides an environment to perform timing-driven partitioning over multiple FPGAs. InCA has an FPGA partitioner named Concept Silicon [18] which partitions an FPGA or PLD netlist onto multiple FPGAs. Quickturn's RPM emulation system [19] creates a hardware prototype from an ASIC or full-custom chip netlist. A hierarchical partitioner is used to partition the design over as many FPGAs as necessary.

3. MOTIVATION FOR A NEW PARTITIONING ALGORITHM

The partitioning of the MFS precedes the subsequent place and route stage of the single FPGAs. The function of the partitioning system in such a top-down flow is to serve as a global placement or floor-planning stage in order to generate a *good* hierarchical solution. Each partition or chip will be transformed into an independent layout problem after partitioning. A partition has a relative position with respect to other partitions. The components assigned to a partition thus acquire a global placement with respect to other components after partitioning. The global connectivity and the relative placement of the components brings in a sense of global distance between components.

The MFS system thus needs a rectilinear partitioning approach. The sub-netlists obtained after partitioning need to be physically assigned to the FPGAs. Minimizing the total wire length between components while partitioning will effectively minimize congestion in addition to reducing the global lengths of nets between partitions. The routability of the *global* connections due to I/O signals and the inter-partition signals can be maximized by minimizing total wire length between partitions and system I/Os. The signal path timing constraints of the system need to be satisfied. The timing violations need to be computed based on the position of the components in the MFS.

Multi-way netlist partitioning strategies like recursive mincut bipartitioning strategies or enhanced mincut for multiple partitions only minimize the nets cut between partitions and do not understand the notion of distance between partitions. These methods thus cannot be deployed successfully to this problem since issues such as total wire length and the length of critical signal paths cannot be controlled. Minimization of the number of pinouts on a partition is just one of the several important objectives of an MFS partitioning system. Furthermore, in this paper we will show that N-way partitioning by recursive application of mincut bi-partitioning yields inferior results compared to our new approach.

The basis for our partitioning algorithm is simulated annealing since it is easy to incorporate signal path timing constraints [23] into the partitioning problem. Initially, this might seem surprising since the computation time for partitioning a flat netlist using simulated annealing could be similar to that of placing the flat netlist, which is precisely what we seek to avoid by using partitioning. In fact, the partitioning can proceed at least an order of magnitude faster than placement. There are several reasons for this. First, in placement a given component can reside at the position of any other component. In other words, a component can be placed in $O(n)$ positions, where n is the number of components. However, in partitioning, the number of possible positions for a given component is on the order of the number of partitions, and this is essentially a constant, independent of the size of the netlist. Hence, in some sense the complexity of the state space is one order lower. Second, in placement it is necessary to get the absolute minimum total wire length since our research has shown that an additional percent or two decrease in wire length tends to yield an additional percent or two of area savings due to a corresponding reduction

in congestion. However, in partitioning, it is not necessary to achieve the absolute minimum total wire length as long as the timing requirements and the pin out constraints are satisfied. Third, we present a new cell clustering algorithm in section 6 which greatly reduces the number of moveable objects (clusters), the number of nets (connections) and the complexity of the nets (the number of clusters per net).

Our MFS partitioning system consists of three main phases as shown in Figure 3. We first reduce the netlist using a new netlist clustering algorithm. The clustered netlist is then partitioned using a simulated-annealing based partitioning algorithm. Finally, chip-level pin assignment is performed using a graph-based global router.

In section 4, we describe the simulated-annealing based partitioning algorithm. The pin assignment stage is described in section 5. The key issues which affect the performance of the simulated annealing algorithm in phase 2 were strongly considered in the design of our new clustering algorithm which is used in the first phase. We thus present the clustering algorithm in section 6. Section 7 is devoted to the results and the conclusion is the subject of section 8.

4. PHASE 2: SIMULATED ANNEALING BASED *N*-WAY PARTITIONING

The MFS partitioning is achieved by a simulated-annealing-based algorithm. This algorithm is a timing-driven *N*-way partitioner which understands the notion of distance between the various partitions. It uses a simplified cost function which consists of the total weighted wire length and a penalty for timing violations. The cost function does not consist of any penalty function involving capacity of partitions analogous to overlap penalty in [7]. Instead, during new state generation, the partitioner only picks moves which are feasible in terms of a pre-defined target utilization. This allows the algorithm to condense the search space of new states and thus improve run time. The system-level pin assignment or pad placement must be performed with respect to direct I/O connectivity of the components and thus must be performed simultaneously during partitioning. The new state generation function picks moves which involve both circuit components and I/O pads in order to accomplish pad placement at the same time as the MFS partitioning. The annealing schedule used is a statistically derived schedule proposed by Lam [22].

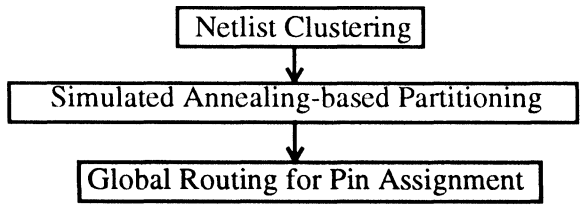

FIGURE 3 The main phases of the MFS partitioner.

Each FPGA chip is considered a partition and the chip edges are the cut lines for partitioning. The number of the cut lines depend on the number, size and the boundaries of the partitions and the size of the MFS. The rectangular regions formed by the intersection of these cut lines are defined as *bins*. If the cut lines do not produce a *satisfactory* grid and thus a *satisfactory* number of bins, additional cut lines are placed automatically to obtain a finer grid. The purpose of dividing the core into bins is to make wire length calculations more accurate. During the partitioning process, a component to be partitioned will move from bin to bin. Its location at any instant is taken to be the center of the bin to which it currently belongs. The finer the grid the cut lines produce, the higher the number of bins they generate. A large number of bins make the wire length calculations more precise, especially with respect to timing. However, with a large number of bins, the search space for component moves is large. This makes annealing more expensive in terms of CPU time. An effective trade-off between the accuracy of wire length and CPU time was obtained by using a number of bins on the order of 20 or the total number of partitions, whichever is greater.

Figure 4 shows an example of bin configurations for an MFS with four FPGAs. The I/O slots shown

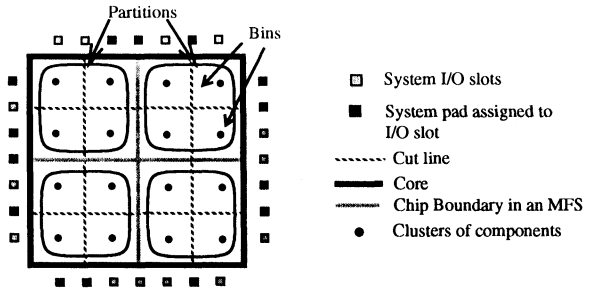

FIGURE 4 Examples of partitions, bins and cut lines.

are for the system-level (top level) pad placement. Note, the chip-level I/O slots (for second level pads) are not shown.

4.1. Cost Function

The partitioner cost function C consists of two terms as shown in (1). The first term is the total weighted wire length, represented by W. The second term is the timing penalty function, represented by P_t.

$$C = W + P_t \tag{1}$$

4.1.1. Total Weighted Wire Length W

At the end of partitioning, its is desirable to obtain the lowest number of pin-outs possible for each chip, since there are a limited number of chip level I/O slots. If each net had the same weight, minimizing the total wire length would not generally minimize the number of pin-outs. We therefore introduced a new dynamic net-weighting scheme which serves to minimize the number of pin-outs. In our scheme, nets which traverse two adjacent bins but are in two different chips (e.g. N_2 in Figure 5) must be penalized more than nets which traverse two bins in a single chip (e.g. N_1). Since one of the major hard-constraints is the number of I/O's available per chip, we formulated a net-weighting scheme which is guided by the number of I/O's a signal needs if it traverse more than one chip. The nets which are restricted to one chip, or the *single-chip* nets, do not need any I/Os and thus have a weight equal to 1.

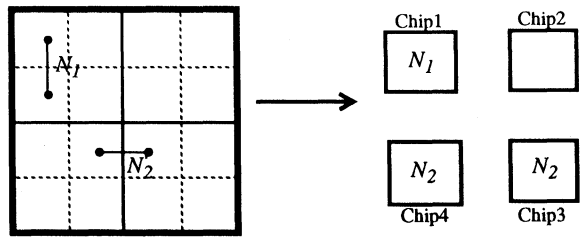

FIGURE 5 Comparing two nets of same length but different weights.

Figure 6a) and 6b) show the conditions when a net traverses two adjacent chips and two diagonal chips, respectively. Both nets need at least two I/Os to make the connection between the two chips. Figure 6c) shows a net which traverses three chips and which needs at least four I/Os. Figure 6d) shows a net which traverses four chips and which needs at least six I/Os. For each net topology encountered in a given MFS, integral weights are designed depending on the number of I/Os it needs for inter-chip connections, IO_n. A constant, K, increases the differences in weight from the *single-chip* nets. The weight of a net, n, is $w_n = 2IO_n + K$. We use $K = 2$. The summation of all the half perimeters of the nets weighted by the dynamic net weight is the total weighted wire length, (as in (2)) for a particular configuration of cells. At any

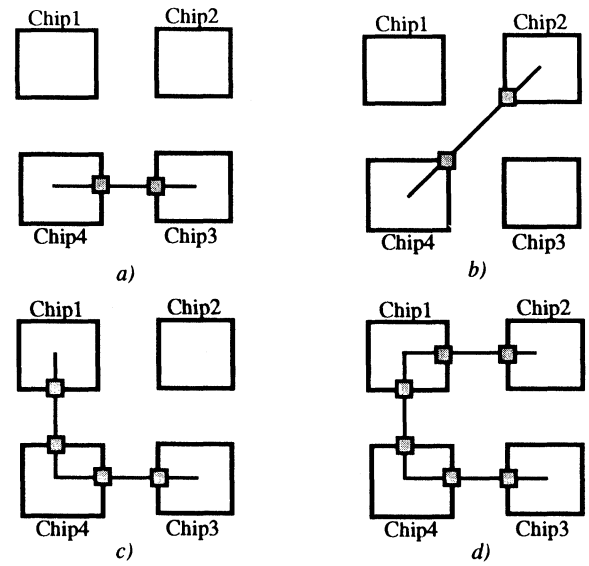

FIGURE 6 The net weighting scheme.

point during the annealing, the new weight of a moved net is re-evaluated and used to compute the weighted wire length. W is given by:

$$W = \sum_{n=1}^{N_n} (S_x(n) + S_y(n)) \bullet w_n, \tag{2}$$

where $S_x(n)$ and $S_y(n)$ are the width and height of the minimum bounding rectangle of the net, respectively, and w_n is the weight of net n. In order to minimize the CPU time necessary to update W for large nets, we use an incremental net-span updating scheme. The incremental scheme devised for the partitioner takes advantage of the gridded nature of the bin structure used and thus is simpler and faster than previously reported methods [20,23]. Since detailed placement is going to follow this partitioning stage, the clusters of components are assigned to the center of the bins and the pins of a component are also taken to be at the center of the component. Unlike placement, where we need the exact location of a pin for precise wire length calculations, for each net the partitioner only needs to store the number of pins for a net at each grid line. Thus, updating the pin configuration of a net after a move is easy. The global scale of the grid lines are stored in a lookup table. The x and y span of a net is calculated using the maximum and minimum of the active grid lines of the net (*i.e.* grids which contain one or more pins for this net).

4.1.2. Timing Penalty

The timing penalty in the cost function is calculated based on the slacks generated in the critical paths of the circuit by partitioning. A critical path may consist of several nets. The timing penalty is minimized dynamically during partitioning. In this section, we will first describe the propagation delay model we have designed for a timing path over multiple FPGAs. Based on this model, we will define the timing penalty.

The total delay on a path p over multiple FPGAs is the sum of the delay generated in the configurable logic blocks (CLB) inside each chip, $T_L(p)$, and the total interconnect delay, $T_R(p)$.

$$T_{pd}(p) = T_L(p) + T_R(p) \tag{3}$$

$T_R(p)$ is the sum of all the constituent net routing delays, $T_R(n)$, due to the intra-chip and inter-chip connections of the net.

$$T_R(p) = \sum_{n \in p} T_R(n) \tag{4}$$

Logic Delay: The total logic delay of a path p is:

$$T_L(p) = N_D \bullet T_{CLB}, \tag{5}$$

where N_D is the number of logic levels or depth of the particular critical path and is available from the logic synthesis stage of the circuit. T_{CLB} is the intrinsic delay of the CLB. For a given technology and CLB design of an FPGA, T_{CLB} is constant and independent of the configuration, number of inputs or outputs. (According to spice simulation results reported by Singh *et al.* in [25], the typical worst case delay of a 4-input lookup-table based logic block in a $\lambda = 1.2$ μm CMOS process is about 1.7ns). Rose *et al.* evaluated and compared the performances of Altera, Actel and Xilinx FPGAs in [26]. We will follow their assumption that for $1\mu m \leq \lambda \leq 1.75\mu m$, the gate delay is approximately proportional to λ. Given a T_{CLB} of a CLB for a particular λ, we accordingly scale T_{CLB} for the same CLB design for a different λ in that range.

Routing Delay: The total routing delay of a net n, $T_R(n)$, is the sum of the delay due to the intra-chip, $T_S(n)$, and inter-chip connections, $T_M(n)$, of a net.

$$T_R(n) = T_S(n) + T_M(n) \tag{6}$$

Intra-chip Routing Delay: $T_S(n)$ is a function of the routing architecture of the FPGAs used, fanout of a connection, length of a connection, the process technology, and the programming technology. The two main components of $T_S(n)$ is delay due to the switches in the interconnect path and the parasitics of

the wire segments. The delay due to the switches can be modeled for a particular programming technology and the number of switching stages between CLBs in the routing architecture as shown in [27]. (For the anti-fuse technology and single segment routing, the number of switches between two logic blocks were taken to be 2 and the RC model was formed accordingly in [27]). The total switching delay including the parasitics seen by the wire segments used by the net can be modeled as a lumped RC:

$$T_S(n) = R_{sw}C_{sw} = R_{sw}(C_g + C_p) \tag{7}$$

R_{sw} is the equivalent drive resistance or the switching ON resistance and C_{sw} is the total load capacitance seen by the driver. C_{sw} consists of the gate input capacitance, C_g, and the parasitic capacitance, C_p, of the wire segments used to form the interconnection. [For an anti-fuse technology (*e.g.* Actel), R_{sw} does not change with λ. However, for pass transistor technology (*e.g.* Xilinx), R_{sw} is proportional to λ. C_g is proportional to λ in both anti-fuse and pass transistor technology.] C_p depends on the process technology used for the wiring segments and can be computed using the lumped capacitance model and is proportional to wire length. The wire length of a net can be estimated at the partitioning stage using the half-perimeter bounding box:

$$C_p = C_{L_h}S_x(n) + C_{L_v}S_y(n) \tag{8}$$

C_{L_v} and C_{L_h} are the capacitances (per unit length) of the vertical and horizontal tracks or busses in the routing architecture. Thus (7) can be expanded as:

$$T_S(n) = (R_{SW}C_g + R_{SW}[C_{L_h}S_x(n) + C_{L_v}S_y(n)]) \tag{9}$$

Inter-chip Routing Delay: In addition to the delay in the FPGA chips, a net acquires an additional delay when it crosses the chip boundaries. Depending on the type of MFS, MCM or PCB, the modeling of an interconnect between two chips will differ [28]. Interconnect wires on PCBs are usually wider (60–100μm) and thicker (30–50μm) than thin-film MCMs (where the wire width is in the range of 10–25μm and thickness 5–8μm). Figure 7 shows a generalized model for the interconnect of an inter-chip connection in an MFS following the macro-model described in [28]. The model consists of a transmitter capacitance, receiver capacitance and a transmission line modeling the wire segment in between them. The capacitor at the driving end, C_D, models the output capacitance of the driver and the pad capacitance of the chip on the MFS, while the capacitor on the receiving end, C_L, consists of the input capacitance of the receiver and the pad capacitance of the receiver. PCB interconnects usually have low resistance per unit length and thus behave like distributed LC transmission lines (lossless). These lines are generally terminated with a resistor that matches the characteristic impedance, Z_O, to avoid reflections. The total resistance of MCM interconnect lines is comparable to the characteristic impedance (which depends on the structural properties of the substrate) and are thus lossy. MCM interconnect lines are usually unterminated [29]. The inductance of the chip-to-MCM bond is assumed to be negligible which is typical for the flip-chip-attached integrated circuits. The line parameters R, L, and C of the MFS interconnect will depend on the material properties such as the dielectric constant of the insulator (ε), the resistivity of the metal (ρ), the permittivity (μ) of free space and the line geometry of the wire.

Based on this model, the delay for a chip-to-chip interconnect, T_{CC}, appropriate to the particular MFS is pre-computed. We assume that a net which connects to more than one chip will be connected by the

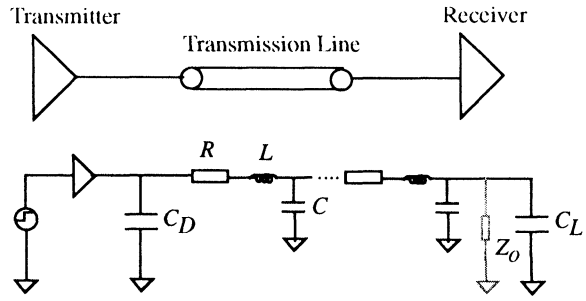

FIGURE 7 The inter-chip interconnect structure and the circuit used in delay modeling.

shortest path tree between the chips and we let IO_n be the number of inter-chip connections a net requires under this assumption. In our main applications, the spacing between the chips in Figure 8 is comprised of pre-wired connections which run perpendicularly to the chip edges. Hence, the total inter-chip connection delay for a net is:

$$T_m(n) = IO_n \bullet T_{CC} \tag{10}$$

Thus, the total routing delay of a net n over multiple FPGAs is:

$$T_R(n) = T_S(n) + T_M(n) = ((R_{SW}C_g + R_{SW}[C_{L_h}S_x(n) + C_{L_v}S_y(n)]) + IO_n \bullet T_{CC}) \tag{11}$$

The total path delay is:

$$T_{pd}(p) = T_L(p) + \left(\sum_{n\epsilon p} ((R_{SW}C_g + R_{SW}[C_{L_h}S_x(n) + C_{L_v}S_y(n)]) + IO_n \bullet T_{CC})\right) \tag{12}$$

In (12), we can precompute the expressions which are independent of wire length and number of inter-chip connections as:

$$K_p = T_L(p) + \sum_{n\epsilon p} (R_{SW}C_g) \tag{13}$$

$$T_{pd}(p) = K_p + \sum_{n\epsilon p} \{(R_{SW}[C_{L_h}S_x(n) + C_{L_v}S_y(n)]) + T_m(n)\} \tag{14}$$

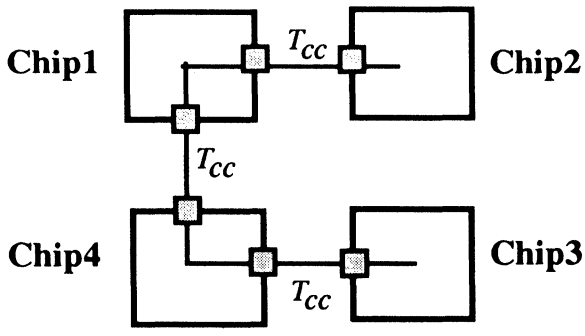

FIGURE 8 Inter-chip connection delay.

As reported in [23], the total timing penalty is computed as the sum of the penalties over all critical paths specified. For each critical timing path, the user supplies an upper bound $T_{ub}(p)$ and a lower bound $T_{ub}(p)$ on the required arrival times. The penalty assigned for a path p is the amount the delay deviates from satisfying the bounds.

$$P(p) = \begin{cases} T_{pd}(p) - T_{ub}(p) & \text{if} \quad T_{pd}(p) > T_{ub}(p) \\ T_{lb}(p) - T_{pd}(p) & \text{if} \quad T_{pd}(p) < T_{lb}(p) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

The total timing penalty is the sum of penalties over all the critical paths specified.

$$P_t = \sum_{p=1}^{N_p} P(p) \tag{16}$$

4.1.3. New State Selection Procedure

An important objective of the partitioner is to insure that each FPGA contains a feasibly implementable amount of logic and to satisfy specific capacity constraints given by the user. The partitioner also needs to perform system-level pin assignment or pad placement on the system-level I/O frame of the MFS. We have implemented a new approach for the generation of new states in the simulated annealing to attain these objectives. (In this section, when we refer to pads or I/Os, we mean system-level pads or system-level I/Os.)

During annealing, the partitioner uses two main types of moves: a single object move and a pair-wise interchange of two objects. An object is either a circuit component or a pad. During a single object move, a component is moved from one bin to another bin or a pad is moved from an I/O slot to another I/O slot. Two objects are inter-changed during a pair-wise interchange move.

The partitioner seeks to maintain an ideal distribution of components in the bins in terms of total utilization of logic per bin. The utilization (U) in a bin is defined as the total component area in that bin di-

vided by the bin area. If the feasible utilization level of the FPGAs is set by the user, the partitioner uses that factor to set the *a priori* target or baseline utilization (U_B) for each bin in those FPGAs. In the absence of such input, the objective of the partitioner is to achieve roughly comparable utilization in each of the bins. The *a priori* target or baseline utilization (U_B) for each bin is defined as the total component area for all of the chips divided by the sum of the chip areas. To obtain the target utilization, a proposed move is only tested through the annealing criterion if it satisfies the *utilization_test*. Basically, a proposed move is evaluated only if the new utilization (U_{new}) does not exceed the baseline value (U_B) or if the new utilization is closer to the baseline than the current utilization. A move passes the *utilization_test* according to the pseudo-code shown in Figure 9. The utilization test acts as a screening routine for the proposed moves. Thus, only feasible partitioning solutions are tested through the annealing criterion.

In order to generate a new state, a component A is selected randomly from the set of all objects: components and pads. If A is a component, we then randomly select a new location within the range limiter window [21]. The bin which covers this location is noted. If the single move of A to this bin passes the utilization test, the proposed new configuration is obtained by attempting a single move to the new bin. Otherwise, a component B is randomly selected from the list of cells assigned to the bin. The pair-wise interchange of A and B is tested through a similar utilization test as that shown in Figure 8. If the test is passed, the proposed new configuration is attempted by pair-wise interchanging the components A and B. On the other hand, if A is a pad, we randomly select an I/O slot to which A can move to. If that I/O slot is empty, a single move of pad A to that slot is attempted. Otherwise, the pad B occupying that slot is trial interchanged with A.

utilization_test (move)
  if $U_{new} \leq U_B$, PASS
  else
    if $|U_{new} - U_B| \leq |U - U_B|$, PASS
    else FAIL

FIGURE 9 Function for utilization test.

5. PHASE 3: GLOBAL ROUTING AND PIN ASSIGNMENT

In this section we will describe the third and final phase of the partitioning system. At the end of annealing, the system-level pads have been placed and each partition contains unplaced components. Following the physical partitioning of the netlist, each of the n partitions (FPGAs) are converted into complete and independent layout problems in this phase. Pin assignment is performed on the chip-level I/O frames so that the chips in the MFS can be interconnected consistently using the pre-wired connections in between chips and those between the chips and system I/Os. This phase is mandatory for MFS partitioning in order to make the application complete. The signals which cross one or more chip boundaries are *external* signals. Given the total number of pin-outs of each partition, the objective is to assign the external signals to the chip level I/O's in a way such that there is no overflow.

We employ a graph-based global router in this phase. An example of the global router graph for an MFS with four FPGAs is shown in Figure 12a. A node is defined at the center of each bin. In order to route nets which connect pad pins, additional nodes are defined outside the MFS core as shown. All rectilinearly adjacent node pairs inside the core are connected by edges. However, to avoid route segments connecting adjacent pads, the edges connecting the nodes which represent pads are excluded from the graph except for the corners of the MFS core. This is done to accommodate pads at those corners (e.g. *PD1* and *PD2*) by providing edges and thus paths to connect to them. A capacity is assigned to each edge. The edges which intersect any chip boundary are assigned a capacity equal to the number of pre-placed interconnect wires or I/O slots available on that boundary within the range of that edge. All internal edges are assigned a large capacity to encourage the router to use these

edges over the external edges if possible. Initially, the weight of an edge is equal to the length of the edge.

The global router seeks the shortest possible routes while minimizing the overflow over available routing resources. The global routing algorithm is shown in Figure 10. Initially, the shortest path routes are found for all external signals. Based on these routes, the total overflow is calculated. The function *update_edge_weight* is used to update the weights of the edges with overflow. The routes which use the edges with overflow are discarded and the corresponding signals are re-routed using an iterative rip-up and re-route scheme. The pseudo-code for generating a route for a net is shown in Figure 11.

Pin assignment is performed based on the final routes given by the global router for the external signals as shown in Figure 12b. For each route obtained for a net, we generate an I/O pin at each intersection of a chip boundary and a route segment. Let the net consisting of pins *pin1*, *pin2* and *pin3* be routed using the L-shaped route as shown in Figure 12a). A pin *a*

```
Set of external nets: N_ex ; Set of edges with overflow E_o ;
Set of routes for net n : R_n
Algorithm Global_Route_for_Pin_Assignment
  for all n ∈ N_ex
      r = generate_a_route( n , 0);
      R_n = R_n ∪ r ; /* add to the set of routes for net n */
  Calculate overflow on all edges and form E_o ;
  Main_iteration = 0;
  While (total overflow > 0 OR Main_iteration ≤ MaxIteration)
      for all e ∈ E_o
          Update_edge_weight( e );
      for all e ∈ E_o
          for all n ∈ e
           r = generate_a_route( n , Max_improve);
           R_n = R_n ∪ r ;/* add to the set of routes for net n */
      Random_interchange(Main_iteration · |N_ex| );
      Increment Main_iteration;
Subroutine Update_edge_weight(edge)
  if overflow(edge) > 0
      weight(edge) = ∞ ;
  else
      weight(edge) = length(edge);
Subroutine Random_Interchange(Rand_Iteration)
  iteration = 0;
  While (iteration < Rand_iteration)
      Randomly pick a net n ∈ N_ex ;
      Randomly pick a route r ∈ R_n ;
      Estimate total overflow with r in place of current route( n
      if (total overflow decreases)
          current route( n ) = r ;
          update E_o ;
  Increment iteration;
```

FIGURE 10 Global routing algorithm.

```
Set of pins for net n : P (n) ; Set of trees for net n : T (n)
Subroutine Generate_a_route(net n, Max_improve)
  Make each node corresponding to pin ∈ P (n) a tree and form
      T (n) ;
  while (|T (n)| > 1 )
      Find a shortest cost path p between two nodes, v_i and v_j
        v_i ∈ T_k and v_j ∈ T_l ;/* T_k ∈ T (n) and T_l ∈ T (n) */
      Merge the trees and path into one tree T_k = T_k + T_l + p ;
        T (n) = T (n) - T_l ;
  if (Max_improve > 0) Improve_route_tree(T (n) , Max_improve);
  return (T (n) );
```

```
Subroutine Improve_route_tree(T, Max_improve)
  iteration = 0;
  while (iteration < Max_improve)
      Select a random edge e ∈ T ;
      Create a path p_1 by tracing e to nodes with degree d > 2 ;
      Create two trees T_1 and T_2 by removing p_1 from T ;
      Find the shortest cost path p_2 between two nodes v_i ∈ T_1
          and v_j ∈ T_2 ;
      If cost(p_2) < cost(p_1) then T = T_1 + T_2 + p_2 ;
      else T = T_1 + T_2 + p_1 ;
       Increment iteration;
```

FIGURE 11 Subroutine generate_a_route for a net.

is created at the left side of *Chip1* and a pin *b* is created at the bottom side of *Chip1*. If such an intersection point is on a side shared by two chips, as in the case of pin *b*, the pin is assigned to both chips. In this case, pin *b* is assigned to both *Chip1* and *Chip2* to maintain consistency in the routing path through that point. Since the routes follow the grid lines, a group of pins are likely to be produced at the same intersection point if several nets share that segment. In such cases, the pins are assigned in the same order on a shared chip boundary. *N* independent layout

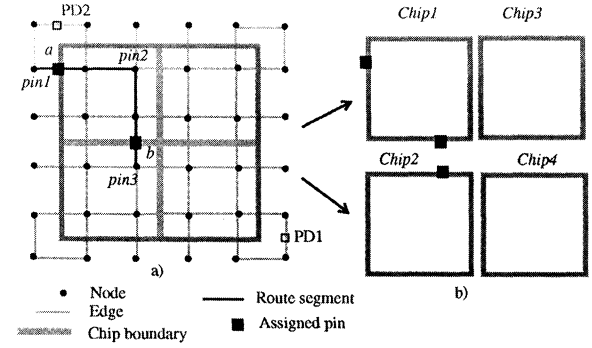

FIGURE 12 a) A global route on the graph. b) Independent chips after pad/pin assignment based on the route.

problems are created at the end of the global routing such that they can be independently placed and routed in parallel if need be.

6. PHASE 1: CLUSTERING

6.1. Motivation for a New Clustering Algorithm

The extent to which a netlist is reduced by clustering depends not only on the characteristics of the original netlist but also strongly on the clustering strategy. The characteristics of the clustered netlist have a strong influence on the quality of results obtained from the partitioner. We have experimented with algorithms using various bottom-up clustering strategies. The effects of the various strategies have helped us to identify the key objectives of a clustering algorithm in the context of our main goal which is to dramatically speed up simulated annealing-based N-way partitioning. The run time of the simulated annealing algorithm implemented in the chip partitioner depends on various factors related to the netlist.

a) *Moveable components*: The total number of components of the MFS which participate in the chip partitioning algorithm determines the total number of moves to be executed in each iteration. A reduction in the number of moveable components will thus improve the run time.
b) *Nets*: When a move is attempted with a component, c, the wire length of the set of nets, N_c, associated with c needs to be updated. The time required to update the wire length of a net $n \in N_c$ is proportional to the cardinality of the fanout of n, $|P_n|$ (P_n is the set of pins on net n). Therefore, the two main factors controlling the run time associated with this move are $|N_c|$ and $|P_n|$. Thus, a component with a large number of nets is an expensive component to move and a net with a large number of pins is an expensive net to update. When components are picked randomly during the annealing process, the larger nets are picked more often than the smaller nets. An objective of the clustering algorithm must therefore be to reduce the fanout of nets as well as to reduce the total number of nets and total number of pins.
c) *Component size*: As described in section 4.1.3, the chip partitioner uses a utilization test routine to maintain a feasible partitioning solution at each step in the annealing process. If two components picked for pairwise interchange differ in their size by a considerable amount so as to fail the utilization screening test, the partitioner goes on to generate another move. Generating infeasible moves wastes CPU time and inhibits full exploration of the state space. Thus, the clustering algorithm should try to generate clusters which are as uniform in size as possible.

Three prominent clustering metrics were proposed recently. Intuitively, the *degree/separation* metric used in the random-walk based clustering algorithm, RW-ST [13], strengthens the objective of seeking a minimum cut between clusters. However, the RW-ST heuristic has overall worst-case time complexity of $O(n^3)$ making it inefficient for its application to large circuits. The shortest path clustering algorithm [12], based on uniform multi-commodity flow problem, was evaluated using the ratio cut metric. In most cases the clusters produced by this method had a large range of size. Clusters having such properties are not at all suitable for N-way partitioning as discussed previously.

A third clustering algorithm based on (k,l) connectivity was reported in [24]. If there are k edge-disjoint paths of length l between components s and t, then s and t are said to be (k, l)-connected. A cluster was defined as a group of components such that every two components in that group are (k, l)-connected directly or indirectly through transitive closure. We have extended this algorithm for multipin nets and have implemented it for evaluation purposes. We have improved the worst case time complexity of this algorithm to $O(nB^2)$ for $l = 2$ and $O(nB^3)$ for $l = 3$. Here n is the number of nodes/components in the graph and B is a constant representing the upper bound of the number of immediate neighbors a component has

in a circuit. However we found several disadvantages with this algorithm. The $(k, l < 1)$ criterion may yield non-intuitive clusters if the circuit is not structured enough. Also, there is no control over the size of the clusters obtained. The algorithm uses the netlist connectivity graph for enumeration of paths. The use of such an algorithm for large circuits is impractical. Through extensive experimentation of this algorithm with $(k, 1)$, combined with other heuristics, we obtained clusters which were of reasonable quality. However, since the appropriate value of k was difficult to predict without experimentation on a particular circuit, we could not use this algorithm to generate clusters for the chip partitioner over a wide range of circuits.

6.2. A Two-Phased Natural and Adaptive Approach for Clustering

Our clustering approach is a bottom-up hierarchical technique based on an agglomerative method of clustering. At each level of hierarchy, we cluster nodes which qualify to merge with each other. This results in a netlist of reduced complexity. Our experiments on circuits having a large size range have showed that the clustering process needs to be adaptive to the current state of the netlist in order to obtain the best N-way partitioning results. Thus we have devised a two-phased natural and adaptive clustering technique. The first method finds the natural clusters of the circuit. The criterion for merging nodes at each level of hierarchy is based on the net connectivity and density of the weighted netlist graph. The second and more adaptive strategy is based on a heuristic technique which aims to refine the netlist obtained from the first method for its most effective application in the chip partitioner.

Since we are interested in circuit applications, we wish to minimize the number of inter-cluster nets. Thus, we considered the k-edge-connectivity of a graph as opposed to k-vertex-connectivity of a graph. Our initial concept of a cluster was derived from the notion of $(k, l = 1)$ connectivity [24]. However, we adopted a more general model of a graph representation for a netlist which allowed us to handle multipin nets as easily as 2-pin nets. The *accumulative weighted graph* is simpler and more accurately represents the circuit structure than a hypergraph. The time complexity of our basic algorithm is linear with respect to the number of nodes in the accumulative weighted graph.

Accumulative Weighted Graph

The nodes of the graph represent the components of the circuit and an edge between two nodes indicates that a hyper-edge contains these two nodes. For example, each multipin net connecting n components is represented by a complete graph contributing $n(n - 1)/2$ edges to the graph. Our ultimate purpose is to use the clusters for distance-based partitioning. In a final placement, the nets with very high fanout naturally span a larger portion of the core and thus have higher wire lengths. Thus, partitioning programs will have more success in minimizing the net lengths of low fan out nets. In order to differentiate between large and small nets, we have formulated an edge weighting scheme based on the fanout of a net. Thus an edge representing a net with n pins is given a weight $1 / n - 1$. An edge from a 2-pin net is given a weight of 1. After assignment of weights, we collapse all the edges between a pair of nodes into one edge. This final edge thus carries an aggregate weight equal to the sum of weights of the original edges between those nodes. In this way, as shown in Figure 13, we can represent the global nature of the connectivity of the netlist through a simplified graph as a pair of nodes will have at most one edge between them.

DEFINITION 1: An edge of the weighted graph with a weight $w > 1.0$, is a *strong edge*.

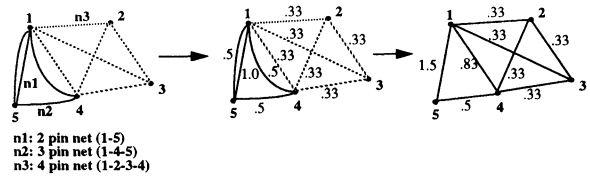

FIGURE 13 Weighted graph derived from a hypergraph.

DEFINITION 2: *Natural cluster*: A group of components such that every two vertices (s, t) are either connected by a *strong edge* directly or indirectly through transitive closure.

Edge (1–5) in Figure 12 is a strong edge.

6.3. Multilevel Cluster Growth

Natural Clustering Algorithm

The purpose of the first phase of clustering is to extract dense subgraphs (*natural clusters*) from the weighted graph of the netlist. The initial seed for this agglomerative growth process consists of the individual components in the netlist. We start out with the netlist of the circuit and construct the weighted graph. Nodes which are connected through *strong edges* are clustered. The *Cluster_Natural* algorithm is shown in Figure 14. When nodes v and v_i are merged using the natural clustering algorithm, the edges which were between them become internal edges to the cluster these two nodes belong to. These internal edges do not participate in subsequent levels of clustering. The external edges get new weights depending on the new accumulative graph obtained after updating the netlist. The clusters are grown in consecutive layers by iterative calls to the Cluster_Natural algorithm. At the second level and below, the *big* clusters already formed at previous levels are not allowed to grow anymore. We have defined *big* as a multiple of the average component size. The process is frozen when no more strong edges are found.

```
Input: V[G] : Set of vertices in the graph G,
       F_e(v) : Set of adjacent edges of v.

Algorithm Cluster_Natural()
    for all vertices v ∈ V[G]
        for all e_i ∈ F_e(v) such that e_i = (v, v_i)
            if ⟨weight(e_i) ≥ 1.0⟩ then
                Cluster(v, v_i)
```

FIGURE 14 Cluster_Natural algorithm.

Adaptive Clustering Algorithm

Since the first phase of clustering is dependent on the density of the graph, the distribution of components in clusters may not be uniform. The clusters obtained in the first phase are thus refined through heuristic adaptive clustering techniques. These techniques were designed to further mold the shape of the reduced netlist to make chip partitioning more efficient. With a minimal number of changes to the natural clusters, it is desirable to further reduce the complexity of the network while trying to achieve a relatively uniform distribution of clusters sizes. We identified two main ways of reducing the size of the network:

a) collapse a small net (typically 2-pin nets) and thus decrease the total number of nets in the network,
b) reduce the fanout of a large net and thus decrease the average net fanout.

A set of *small* clusters is formed and used in the algorithm *Cluster_Adaptive()* (Figure 15). A *small* cluster is defined as a cluster with size less than the average cluster size.

We tested this two-phased natural and adaptive approach of clustering on several MFS circuits. The nature of the clusters obtained on one such circuit is shown in Figure 16. The number of clusters containing a particular count of components and the aggregate cluster area (sum of the constituent component areas) versus the number of components in a cluster are shown. Note that even though the number of

```
Input: V_I[G] = Set of vertices corresponding to small clusters

Algorithm Cluster_Adaptive()
    for all vertices v ∈ V_I[G]
    do merge (v)

Subroutine merge (v)
    If (v has a two pin net n) then
        pick cluster c on net n such that c ≠ v AND c ∈ V_I[G]
        if (c exists) then
            Cluster(v, c)
    else
        create N(v) , the set of nets containing v
        let N_s(v) ⊆ N(v) be the set of nets having the highest number of pins
        create set C of the clusters connected to nets in N_s(v)
        pick smallest cluster c ∈ C
        Cluster(v, c)
```

FIGURE 15 Cluster_Adaptive algorithm.

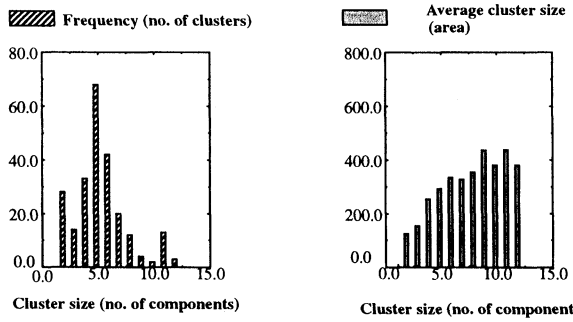

FIGURE 16 Distribution of MFS circuit components in clusters.

components per cluster has a wide range, the average size (total component area) of the clusters is relatively uniform.

Our algorithm was designed to be an efficient implementation and thus usable for practical applications. Efficient data-structures like disjoint-union sets are used to form clusters. The worst case time complexity of our algorithm for multipin nets is $O(nB)$, where B is a constant representing the upper bound of the number of immediate neighbors a cell has in a circuit and n is the number of cells. Due to the accumulative weighted graph, the space requirements for our algorithm are $O(n^2)$ in the worst case. Practical circuits generate much sparser graphs than complete graphs and thus require approximately $O(n)$ space. Also, this space requirement is for the first level of clustering only. As the network is reduced in successive levels, the space requirement reduces accordingly.

7. RESULTS

Our MFS partitioning system, *Tomus*, has been developed in C with an X11 graphics interface to provide interactive features. Several industrial circuits were used to test the partitioning system. The circuit descriptions are shown in Table I. Figure 17 shows the partitioning results of the circuit **dma** for a 2-FPGA MFS. The net connectivity between bins is shown before and after partitioning. This demonstrates the effectiveness of *Tomus* in minimizing congestion across chip boundaries.

Table I Circuit descriptions.

Circuit	Midi	Ensm	Decimate	Adpcm	Dma	Mha
No. of Macros	525	936	958	1595	1399	1848
No. of CLBs	884	1642	1803	2847	1615	1718
No. of System IOs	61	77	65	62	106	89
No. of Nets	689	1053	986	1904	1450	1799
No. of pins	2170	2700	3267	5151	3909	5162
Average fanout	2.51	2.56	2.89	2.7	3.03	3.17

Partitioning results using clustering

The reduced netlist obtained from our clustering algorithm was used during the normal course of the MFS partitioning. Once the MFS partitioning is complete, the clusters are disintegrated into the original components. A low temperature annealing is performed on these components using only pairwise interchange of components. Since updating the cost function in annealing is the major speed bottle-neck, a reduced number of nets and pins speeds up the annealing process compared to using a flat netlist. Table II compares the partitioning results between the flat mode and the clustered mode of *Tomus* on a 4-FPGA MFS. For a total of 20 runs of *Tomus*, we report the minimum of the nets cut, the average and the minimum overflow (unroutable nets) after chip-level pin assignment. The CPU times reported are on a DEC 5000/200. For the clustered mode, the CPU time includes the time taken to generate clusters. On average we obtained a speed up of up to 5 times while improving the quality of the partitioning results in all cases as shown in Table II. Based on the encouraging results obtained using clusters, we have used the clustered mode of *Tomus* for rest of our experiments in the following sections.

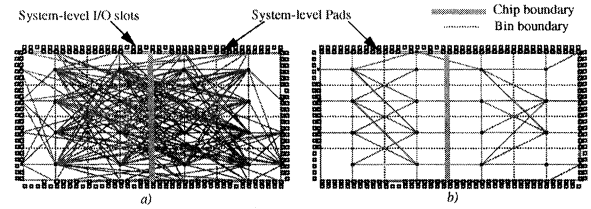

FIGURE 17 Partitioning results of dma on a 2-FPGA MFS, a) before and b) after partitioning.

Table II Improved partitioning results using clustering.

Circuit	Flat				Clustered				Reduction	
	Nets Cut	Overflow		CPU(s)	Nets Cut	Overflow		CPU(s)	Overflow	CPU
	Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.		
Midi	17	0.15	0	953	14	0.05	0	250	—	3.8X
Ensm	62	0.2	0	2304	42	0.15	0	442	—	5.2X
Decimate	96	12	4	2723	96	5.95	0	914	100%	3.0X
Adpcm	114	20.2	12	5877	105	13.6	9	1382	25%	4.25X
Dma	83	10	7	3843	79	4.65	0	1000	100%	3.8X
Mha	96	18.6	10	5573	94	6.5	0	1341	100%	4.15X

Global routing

In this section, we show results of the pin assignment phase in *Tomus*. We partitioned each circuit using a 4-FPGA MFS. Given the total number of nets cut over the four FPGAs at the end of the simulated annealing phase, global routing was executed for pin assignment. The global router first finds the shortest routes. An iterative rip-up and reroute process then improves these routes to minimize overflow. In Table III, out of a total of 20 runs, we picked the run which gives the best final overflow and compared the overflow with the initial shortest routes and the final overflow after overflow minimization. In all but one case, the overflow (unroutable nets) was reduced to zero. For the case with non-zero overflow, the routing was completed by using unassigned system-level I/Os (which are not available for most designs).

Comparison with Mincut

In this section we compare the N-way partitioning results using *Tomus* with results from recursive bi-partitioning using mincut. We have implemented the mincut bi-partitioning algorithm proposed by Fiduccia and Mattheyses [5].

Table III Overflow minimization using global routing.

Circuit	Nets cut	overflow (shortest routes)	overflow (after overflow minimization)
Midi	14	0	0
Ensm	42	0	0
Decimate	96	7	0
Adpcm	105	12	9
Dma	79	3	0
Mha	94	5	0

Four-Way Partitioning

For an MFS with four FPGAs, we partitioned each circuit recursively applying Fiduccia and Mattheyses (F–M). Table IV shows the 4-way partitioning results using recursive mincut. For each run of F–M, we start with different initial partitions. Over a total of 30 runs on each circuit, we report the best results in terms of nets cut in Table IV. We used *Tomus* to physically partition each circuit onto four FPGAs. Table V shows the results using *Tomus*. Equal partition sizes were specified in both cases. The best results of 30 runs are shown for both F–M and *Tomus*. For all the circuits, *Tomus* obtains lower or equal number of nets cut between four FPGAs. The average improvement of *Tomus* over recursive F–M is 34%.

Table IV F-M Mincut results for four-way partitioning.

Circuit	Midi	Ensm	Decimate	Adpcm	Dma	Mha
No. of CLBs in FPGA1	172	375	413	746	321	453
No. of CLBs in FPGA2	201	356	471	651	337	408
No. of CLBs in FPGA3	261	443	458	732	537	433
No. of CLBs in FPGA4	250	468	461	718	420	424
Nets Cut	33	95	96	155	109	151

Table V Tomus results for four-way partitioning.

Circuit	Midi	Ensm	Decimate	Adpcm	Dma	Mha
No. of CLBs in FPGA1	240	374	474	718	435	363
No. of CLBs in FPGA2	203	450	497	784	344	461
No. of CLBs in FPGA3	244	427	423	765	404	429
No. of CLBs in FPGA4	197	391	409	580	432	465
Nets Cut	14(−57%)	42(−56%)	96(0%)	105(−32%)	79(−27%)	91(−39%)

Two-Way Partitioning

We used another MFS with 2 FPGAs and conducted a similar experiment. For each run of F–M, we start with different initial partitions. Over a total of 30 runs on each circuit, we report the best results in terms of nets cut in Table VI for 2-way partitioning results. The best results of *Tomus* for 2-way partitioning from 30 runs per circuit are shown in Table VII. Again, for all circuits, *Tomus* obtains lower nets cut. The average improvement of *Tomus* over F–M is 40%.

Comparison with InCA

We have also compared the *Tomus* partitioning results for a 4-FPGA MFS with the test results from an industrial FPGA partitioner (*Concept silicon* from InCA). The overflow (number of unroutable nets) after pin assignment is compared in Table VIII. The average improvement of overflow is 73%.

Timing Results

We tested the timing-driven capabilities of the partitioner on the MCNC benchmark circuits. Multiple FPGA configurations using Xilinx devices on a PCB were used to obtain these results. For each circuit we first used *Tomus* to find a partitioning without imposing any delay bounds. Using the nominal net lengths obtained from this run, we extracted (in order) the m most delay critical primary input (PI) to primary output (PO) pin pairs. (The value of m was limited so as to not more than double the overall CPU time versus the case when no delay bounds are imposed. We verified that none of the non-included pin pairs gave rise to a critical delay at the conclusion of the partitioning).

Table VI F-M Mincut results for two-way partitioning.

Circuit	Midi	Ensm	Decimate	Adpcm	Dma	Mha
No. of CLBs in FPGA1	441	821	900	1425	810	859
No. of CLBs in FPGA2	443	821	903	1422	805	859
Nets Cut	38	81	83	67	69	128

We extract the current longest path between these pins. These constitute the set of critical paths used in our timing penalty function, and we impose the delay bound on these paths. Because a particular critical path may not always be the critical path for a pair of pins, we update the set of critical paths 150 times (once per iteration) during the course of the annealing based partitioning.

We compared the results with the timing penalty deactivated versus the results obtained with the timing penalty activated for each circuit. In Table IX we have shown the number of paths which were within specifications and the number of paths which were outside the specifications in both cases. Column 1 shows the number of devices used. Using our timing penalty function, *Tomus* successfully partitioned these circuits satisfying the timing constraints in all cases. In all the circuits, *Tomus* achieved a significant reduction in the longest path delay by using the timing penalty function. The average reduction was 17%. These results were obtained at the cost of 17% in nets cut and 5% in wire length on average.

8. CONCLUSIONS

We have presented a timing driven N-way partitioner for MFSs. The physical constraints of the MFSs and

Table VII Tomus results for two-way partitioning.

Circuit	Midi	Ensm	Decimate	Adpcm	Dma	Mha
No. of CLBs in FPGA1	496	766	925	1407	833	830
No. of CLBs in FPGA2	388	876	878	1440	782	888
Nets Cut	15(−60%)	40(−50%)	64(−22%)	27(−59%)	44(−36%)	106(−17%)

Table VIII Comparison with InCA.

Circuit	InCA (Industrial Tool)	*Tomus*	Improvement
Midi	0	0	—
Ensm	16	0	100%
Decimate	28	0	100%
Adpcm	15	9	40%
Dma	73	0	100%
Mha	53	0	100%

the timing constraints are satisfied during partitioning. The partitioning algorithm minimizes total weighted wire length in order to minimize the pin-outs of each FPGA. We have introduced a timing model which is peculiar to a multiple FPGA architecture. It combines all the possible delay factors involved in a system with multiple FPGA-based chips. The pin assignment phase is mandatory for MFS partitioning in order to make the application complete. Although the global router uses a few well known heuristics, however, we have made appreciable extensions which can handle the pre-routed connections of an MFS structure. We have introduced a new two-phased natural and adaptive clustering algorithm which has improved the quality and run time of MFS partitioning. The annealing-based N-way partitioner executes four times faster on average using the clusters as opposed to the flat netlist with improved partitioning results. For several industrial circuits, our approach outperforms the recursive mincut bi-partitioning algorithm by 35% on average in nets cut. Our approach also outperforms an industrial FPGA partitioner by 73% on average in overflow. We have tested the timing-driven capabilities of the partitioner. Using the timing penalty function, *Tomus* successfully partitioned several MCNC benchmarks satisfying the timing constraints in all cases. An average reduction of 17% was achieved in the longest path delay at the cost of 17% in nets cut and 5% in wire length on average. In the future, we will extend our research to accommodate new MFS designs with new interconnect wiring designs [1][30].

Acknowledgments

This research was funded by the Semiconductor Research Corporation, Intel Corporation, Digital Equipment, National Semiconductor, and Cadence Design

Table IX Timing driven partitioning.

Circuits	#FPGAs used	# Critical PI/PO pairs	No constraints		With constraints		Increase (nets cut) %	Increase (wire length) %	Reduction (longest path) %
			Within Spec.	Outside Spec.	Within Spec.	Outside Spec.			
c1355	2	100	100	0	100	0	5	3.4	30
c1908	2	100	53	47	100	0	22	4.5	21
c2670	4	100	97	3	100	0	3	3.7	7
c3540	3	81	50	31	81	0	16	1	10
c5315	4	540	32	508	540	0	8.6	1.6	30
c7552	4	450	108	342	450	0	26	12	32
c1196	2	110	28	82	110	0	17	3.2	19
c1423	2	40	24	16	40	0	23	8.6	12.5
c1238	2	67	46	21	67	0	24	5.6	3
c6288	6	81	53	28	81	0	23	2.6	4

Systems. Kalapi Roy would like to thank Bill Swartz, Richard Weier, and David B. Guan for their encouragement and help.

References

[1] P. K. Chan, M. Schlag and M. Martin, "BORG: A Reconfigurable Prototyping Board for FPGAs," *FPGA 92, First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992, pp. 47–51.

[2] D. Thomae, T. Petersen, and D. E. Van den Bout, "The Anyboard Rapid Prototyping Environment," in Carlo H. Sequin, editor, *Advanced Research in VLSI*, pp. 356–370, MIT Press, 1991.

[3] W. O. Mcdermith, "A Bottom-Up Approach to FPGA Partitioning," in *Proc. IEEE Custom Integrated Circuit Conference*, 1992, pp. 5.4.1–5.4.4.

[4] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal* 49/2, 1970, 291–307.

[5] C. Fiduccia and R. Mattheyses, "A Linear time Heuristic for Improving Network Partitions," in *Proc. 19th Design Automation Conf.*, 1982, pp. 175–181.

[6] C. Sechen and D. Chen, "An Improved Objective Function For Mincut Circuit Partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 410–420.

[7] T. K Ng, J. Oldfield, and V. Pitchumani, "Improvements of a Mincut Partition Algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 470–473.

[8] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks," *IEEE Transactions on Computers* 33/5, 1984, pp. 438–446.

[9] P. Stravers, "Partitioning a Network into *n* Pieces with a Time-Coefficient Net Cost Function", in *Proc. IEEE European Conf. on Design Automation*, 1991, pp. 177–182.

[10] C. Kring and A. R. Newton, "A Cell-Replicating Approach to Mincut-Based Circuit Partitioning", in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 2–5.

[11] Y. C. Wei and C. K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Transactions on Computer-Aided Design*, July 1991, pp. 911–921.

[12] C. W. Yeh and C. K. Cheng, "A Probabilistic Multicommodity-Flow Solution to Circuit Clustering Problems", in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992, pp. 428–431.

[13] L. Hagen and A. B. Kahng, "A New Approach to Effective Circuit Clustering", in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992, pp. 422–427.

[14] G. Vijayan, "Min-Cost Partitioning on a Tree Structure and Applications," in *Proc. Of Design Automation Conf.*, 1989, pp. 771–778.

[15] S. Mayrhofer and U. Lauther, "Congestion-Driven Placement Using a New Multi-Partitioning Heuristic," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 332–335.

[16] M. Shih, E. Kuh, and R. S. Tsay, "System Partitioning for Multi-chip modules under Timing and Capacity Constraints," in *Proc. IEEE Multi-chip Module Conference*, 1992, pp. 123–126.

[17] K. Perry, "Eliminating Barriers to FPGA use by Timing Driven Partitioning," *Electronic Engineering*, Jan. 1993, pp. 41–44.

[18] "Concept Silicon Partitions your Design onto Multiple FPGAs," *Integrated Circuit Applications Pamphlet*, InCA Inc., Campbell, CA 95008, 1992.

[19] S. Walters, "Computer-aided Prototyping for ASIC-based Systems," *IEEE Design and Test of Computers*, June 1991, pp. 4–10.

[20] C. Sechen and K. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 478–481.

[21] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, 1988.

[22] J. Lam and J. M. Delosme, "Performance of a New Annealing Schedule," in *Proc. 25th Design Automation*, 1988, pp. 306–311.

[23] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 336–339.

[24] J. Garbers, H. J. Promel, and A. Steger, "Finding Clusters in VLSI circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 520–523.

[25] S. Singh *et al.*, "Optimization of Field Programmable Gate Array Logic Block Architecture for Speed," in *Proc. IEEE Custom Integrated Circuit Conference*, 1991, pp. 6.1.1–6.1.6.

[26] J. M. Vuillamy *et al.*, "Performance Evaluation and Enhancement of FPGAs," in W. Moore and W. Luk, editors, *FPGAs*, pp. 137–146, Abingdon EE&CS Books, Abingdon, England, 1991.

[27] J. L. Kouloheris and A. E. Gamal, "FPGA Performance versus Cell Granularity," in *Proc. IEEE Custom Integrated Circuit Conference*, 1991, pp. 6.2.1–6.2.4.

[28] A. I. Kayssi and K. A. Sakallah, "Delay Macromodels for Point-to-Point MCM Interconnections," in *Proc. IEEE Multi-chip Module Conference*, 1992, pp. 79–82.

[29] C. W. Ho *et al.*, "The Thin-film Module as a High Performance Semiconductor Package," in *IBM J. Res. Develop*, **26**, 1987, pp. 286.

[30] I. Dobbelaere, "Peripheral Circuit Design for Field Programmable MCM Systems," in *Proc. IEEE Multi-chip Module Conference*, 1992, pp. 119–22.

Authors' Biographies

Kalapi Roy received the B.Tech degree in electrical engineering from the Indian Institute of Technology, Kharagpur, and M.S. and M.Phil degrees from Temple University and Yale University. She received her Ph.D degree from University of Washington in June 1994 and is currently working in the Physical Design Group at Cascade Design Automation. She has developed and supported several tools in the TimberWolf automatic placement and routing tools since September 1989. She spent the summers of 1990-1992 with the corporate CAD group at National Semiconductor Corporation, Santa Clara. Her research interests are

in performance driven multi-way partitioning of integrated circuits, design of FPGAs and multiple FPGA systems.

Carl Sechen received the B.E.E. degree from Minnesota and the M.S. degree from M.I.T. In 1987 he received the Ph.D. degree from UC Berkeley. From July 1986 through June 1992 he was an Assistant and then an Associate Professor at Yale. Since July 1992 he has been an Associate Professor in the Department of Electrical Engineering at the University of Washington. His primary research interests are the design and computer-aided design of analog and digital integrated circuits. He is the principal developer of the TimberWolf placement and routing package. He is a consultant for DEC, Intel, National Semiconductor, AMD, IBM, and Cadence.