VLSI DESIGN
1996, Vol. 4, No. 4, pp. 293–307
Reprints available directly from the publisher
Photocopying permitted by license only

A Sea-of-Gates Style FPGA Placement Algorithm

KALAPI ROY, BINGZHONG (DAVID) GUAN and CARL SECHEN

Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195

Field Programmable Gate Arrays (FPGAs) have a pre-defined chip boundary with fixed cell locations and routing resources. Placement objectives for flexible architectures (*e.g.*, the standard cell design style) such as minimization of chip area do not reflect the primary placement goals for FPGAs. For FPGAs, the layout tools must seek 100% routability within the architectural constraints. Routability and congestion estimates must be made directly based on the demand and availability of routing resources for detailed routing of the particular FPGA. We present a hierarchical placement approach consisting of two phases: a global placement phase followed by a detailed placement phase. The global placement phase minimizes congestion estimates of the global routing regions and satisfies all constraints at a coarser level. The detailed placer seeks to maximize the routability of the FPGA by considering factors which cause congestion at the detailed routing level and to precisely satisfy all of the constraints. Despite having limited knowledge about the gate level architectural details, we have achieved a 90% reduction in the number of unrouted nets in comparison to an industrial tool (the only other tool) developed specifically for this architecture.

Keywords: Field Programmable Gate Arrays, placement, routability

1. INTRODUCTION

With the advent of new Field Programmable Gate Arrays (FPGAs), additional challenges and constraints are imposed on placement and routing tools. Conventional layout tools have been optimized for flexible architectures such as standard cells or gate arrays. In a row-based design, the cells are not restricted to fixed cell locations and thus are free to be placed anywhere in a row. The chip boundary of such designs is not pre-defined before layout. The chip area is determined after placement and routing is executed on the design. The routing tools are free to expand or contract the routing regions estimated by the placement tool if needed. The placement tool thus is not required to estimate the exact amount of routing resources required for a particular placement configuration. For performance critical designs, the placement tools are required to satisfy the timing constraints.

The primary objective of these prior placement and routing algorithms is to minimize chip area. These tools are required to ensure 100% wirability and to meet the timing requirements. Due to the computational complexity of accurately estimating chip area at the placement stage, the wire length metric is conventionally used [1] in the cost function for placement algorithms. Two important goals of placement algorithms are achieved by minimizing the total wire length of semi-custom designs (like standard cells):

1. Minimization of chip area
2. Minimization of congestion in the routing regions.

These goals do not reflect the primary goals of the placement algorithms for FPGA technologies. FPGAs have a pre-defined chip boundary with fixed cell locations and routing resources. Instead of minimizing the chip area, the placement tool needs to assign the logic blocks of the circuit to the given cell locations such that the design is 100% routable. In this process the tool is required to maximize the utilization of logic within the architectural constraints of the FPGA. Traditional objectives like meeting timing requirements and minimizing congestion are as important as for other technologies.

For FPGAs, if the number of routing segments required for a particular placement configuration of an FPGA exceeds the number available, the placement is invalid, however dense and compact it may be. The routability of a net is not a direct function of its length and the congestion of its routing region. The routing architecture guides the routing pattern of a net depending on the availability of resources in the regions the net segments traverse.

Attempts at modifying the conventional placement algorithms to meet the new constraints of FPGAs are certainly possible and are used in the absence of new tools. Initial placement using TimberWolfSC is iteratively improved to meet FPGA constraints in [2]. The subject of routing channels in standard cell or gate-array designs is well studied. Such knowledge can be used to design tools for row-based FPGAs as in [2]. FPGAs which are array-based (Xilinx) or based on the sea-of-gates style (CLI) have practically no similarities with row-based semi-custom designs. Moreover, timing constraints are the only hard constraints handled by tools such as [1]. Additional constraints specific to FPGAs cannot be handled. These tools produce good results given a minimum of constraints and thus are not suitable for the placement of FPGAs.

Several attempts have been made to combine logic synthesis with placement and routing in order to generate more routable designs [3,4]. Routing directed placement has been proposed [5] for the TRIPTYCH FPGA. An integrated placement and routing approach was proposed for the LABYRINTH FPGA [6]. Although current FPGA design styles have some similarities, they differ from one another in their logic block design and routing architectures. Placement and routing tools have to be specifically designed to handle each architecture separately.

The focus of this paper is on the placement algorithms which are aimed at a sea-of-gates style FPGA architecture. Currently, an industrial tool (the only such tool available) is being used for placement and routing of this particular FPGA. This tool has direct access to the gate level architectural details of the logic circuit. We have developed our placement strategy (SOGFP, Sea-Of-Gates FPGA Placement program) which operates at the macro level of the netlist and currently does not make use of the gate level architectural details of each macro. The knowledge of the basic routing architecture and the description of the macros which consist of the basic logic cells are used. The limitations of the basic routing architecture, the fixed layout of the chip structure with its I/Os, and the routing convention of the clock/reset signals impose multiple constraints on the placement tool. With a minimum of knowledge about the gate level architectural details our approach has significantly outperformed the existing industrial tool.

The paper is organized as follows. Section 2 describes the particular architecture this paper focuses on and discusses the layout challenges posed to a place-and-route tool. Section 3 describes our two-phased placement approach. Finally, in section 4 we test our program on several industrial circuits and compare its results with those of the only commercial placement tool developed for this architecture.

Depending on the complexity and the connectivity of a logic block in an FPGA, the routing structure must be designed so that it offers a maximum of flexibility for its interconnections in order to achieve high density placements. If the number of routing segments required for a particular placement configuration of an FPGA exceeds the number available, the placement is invalid, however dense and compact it may be. The placement tools are no longer required to pack cells as dense as possible without considering the routing estimation for such a placement; the limits of the routing architecture must be taken as the foremost constraint.

2. A SEA-OF-GATES STYLE FPGA ARCHITECTURE

The FPGA architecture (Figure 1) we are addressing consists of a symmetrical two-dimensional array of *blocks*. Each *block* consists of a symmetrical ($m \times m$) array of identical *cells*, where m is typically equal to 8. The total logic capacity of this FPGA is about 5000 equivalent gates. A large variety of logic can be configured in each *cell* either as a single cell *macro* or as a larger *macro* which consists of a pre-routed group of cells.

2.1. Routing Architecture

The routing resources consist of three types of connections. First, a cell can be directly connected to its neighboring cells by fixed wires in the North, South, East and West directions. Second, there are *buses* (*local* and *express*, in Figure 2) which traverse the span of a block and are connected through repeaters for long distance connections. Third, a programmable logic cell can also be configured as a *routing cell*, for example, as a through wire or a fanout junction (see *Cell C* in Figure 2).

For direct connections to adjacent cells, there are two sets of connections on all four sides of the cell. One wire in each set is for an input connection to the cell and the other is for an output from the cell. However, depending on the cell configuration, the choice of these direct connections could be limited. The inputs and the outputs of a particular logic block configured in a macro can be configured in only a limited number of ways, thus limiting the usage of all possible paths in or out of the cell.

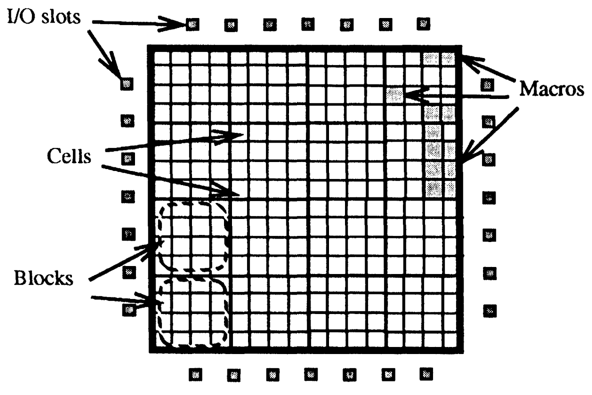

FIGURE 1 A simplified schematic of the FPGA architecture, where $m = 4$.

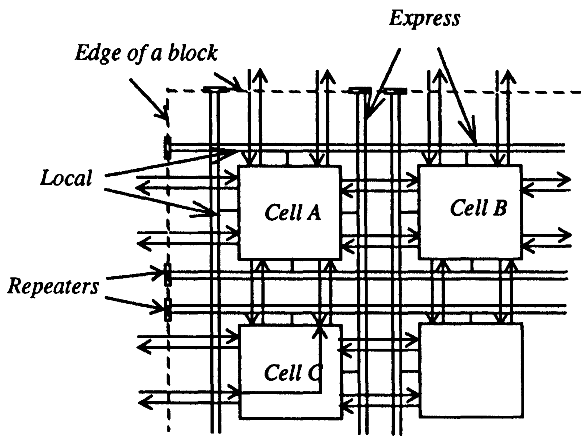

FIGURE 2 The routing architecture of our problem.

A cell has four *local* buses running parallel to each of its edges. On each side of the cell there is an exit to the local bus (see *Cell A* in Figure 2). These local buses are only a block long. But there are repeaters at the terminals of these buses to connect them into a longer routing segment if needed. Thus there are two local buses per column and two local buses per row of cells in each block. In addition there is a pair of *express* buses in each column and row of cells running next to the local buses. These buses have no direct access from the cells. Thus, express buses are efficient and faster for longer net connections over the entire core whereas the local bus connections are suitable for long net connections within a single block. At the junction of two blocks there are repeaters which can be used either to connect segments of only local buses or only express buses together for a longer route segment or to connect a local bus and an express bus for a straight connection or a "U" connection. These connections provide signal regeneration and are thus unidirectional. Bidirectional connections can also be achieved by augmenting the repeater with appropriate logic.

2.2. Clocks and Resets

The top edge of the array has the logic for the distribution of the clock signals and the bottom edge of the array has the logic for distribution of the reset signals. The layout tool must route the clock and reset sources to the top and the bottom edge of the core, respectively (Figure 3). Any single-cell macro or cells of a larger macro which are placed in a column that a clock source feeds can be clocked implicitly by that clock signal. A similar situation is true for resets. Each column can be fed by only one clock and one reset. Depending on the number of clocks and resets in the circuit, the placement of the cells is restricted by the rules posed by the clock and reset arrangements. Since clocks/resets fed to a column are available throughout the whole column, register intensive circuits are easy to configure in this FPGA architecture.

2.3. I/O Pins

There is a fixed I/O frame available to configure I/O pads in this architecture. The slots for I/O's are equally distributed on all four sides of the core (Figure 1). Two adjacent columns/rows cater to one I/O slot, each column/row having routing resources associated with an input buffer for the I/O and an output buffer for the I/O, respectively.

2.4. Limitations of the Routing Architecture

Every cell in this FPGA has pins: AI, BI, AO, BO, and L (Figure 4). Pins AI, BI are input pins; AO, BO are output pins; and pin L can be configured as either input or output pin. If L is configured as an input pin, a cell can be configured as a logic block with at most 3-inputs and 2-outputs. If L is configured as an output pin, a configurable logic block can have at most 2-inputs and 3-outputs. The inputs and the outputs of a particular logic cell can be configured only in a limited number of ways, thus limiting the usage of all possible paths in or out of the cell. Depending on the configuration of a programmable logic cell, only a subset of the input and output pins may be available for routing.

Sense of direction: While routing the source of a signal to its sinks, if a net segment is assigned to a bus, the bus acquires a direction from the source to the sink (Figure 5). Thus, given the positions of the input pins of the sinks, routing resources are assigned such that they satisfy this direction and that the source successfully accesses the appropriate sink input pins. There are at most four local buses available to a cell. However, there may not be any bus available for the particular direction and side a pin requires access. For such a case, the pin can be routed

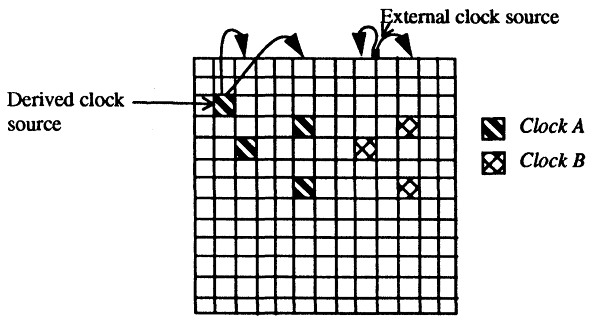

FIGURE 3 Examples of clock nets.

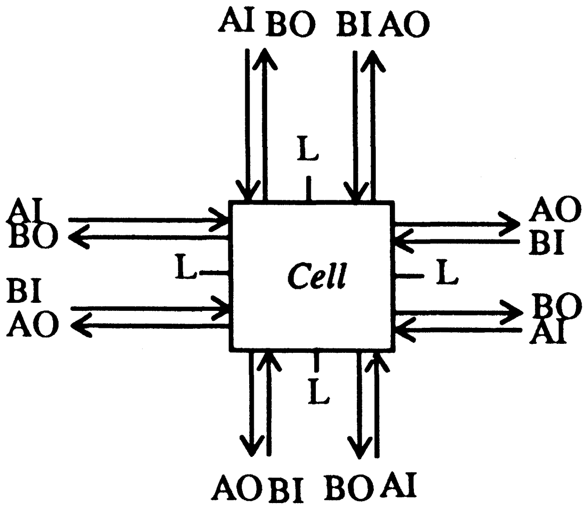

FIGURE 4 Pin configurations of a cell.

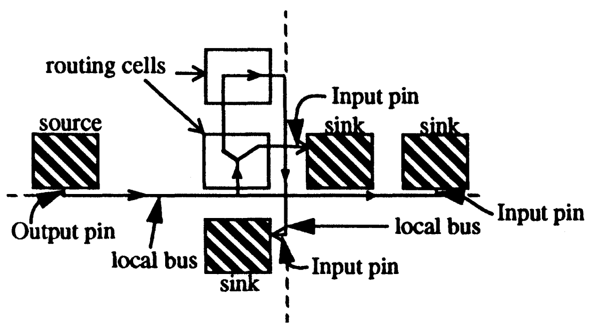

FIGURE 5 A route segment of a signal illustrating the sense of direction.

Figure 6 shows the scheme to route the net which comes out of pin A of cell C1, and goes into pin B of cell C2. If there is no empty cell in between, we will not be able to route the net as shown in the Figure 6a. Only when there is a routing cell R1, which we configure to take input from pin AI and send output to pin BO, can we route the net from pin A of cell C1 and pin B of cell C2 (Figure 6b). Thus for certain direct connections between neighboring cells, availability of routing cells aids the routability. These situations suggest that the placement program should separate the macros as much as possible so that routability of a placement is maximized.

to a nonadjacent bus if there is a path to it by means of a routing cell (*i.e.* unused logic cell). These limitations lead to the usage of a considerable amount of logic cells configured as routing cells around the macros as additional routing resources as shown in Figure 5. Adjacent cells which have common signals can be connected using the direct wires between the cells. However, this is not possible if the corresponding pins of such a signal are not available on the side between the cells. Also, if the direction of the signal to be routed conflicts on the direct connection, an alternate path has to be found possibly using other routing resources like a bus or an empty cell configured as a routing cell.

Absence of switch boxes: Unlike the array style of FPGAs, *e.g.* Xilinx [7], there are no switch boxes available to connect tracks/buses. As analyzed in [8], the flexibility of the switch box design has a great impact on the routability of an FPGA. The switch box provides the resources for a route to change directions from a vertical track/bus onto a horizontal track/bus or vice versa. In the absence of switch boxes, any change of direction in a route will have to made at the cost of other available routing resources. In this sea-of-gates style FPGA, a change of routing direction is typically implemented using routing cells as shown in Figure 5. As a result, utilization levels of this FPGA fall far below typical utilization levels of leading architectures like Xilinx.

3. TWO-PHASED HIERARCHICAL PLACEMENT

The placement tool's primary objective is to determine the best position for each macro and I/O pad such that the routability of the placement is maximized within the following architectural and design constraints:

a) bounded array structure of the chip, b) fixed routing resources, c) clock and reset constraints, d) fixed pad frame and the routing convention of external I/O signals, and e) signal path timing constraints.

The task of handling all of the above constraints and maximizing routability cannot be handled by conventional macro/standard cell placement tools. With multiple FPGA constraints, the problem needs to be solved by means of a divide-and-conquer strat-

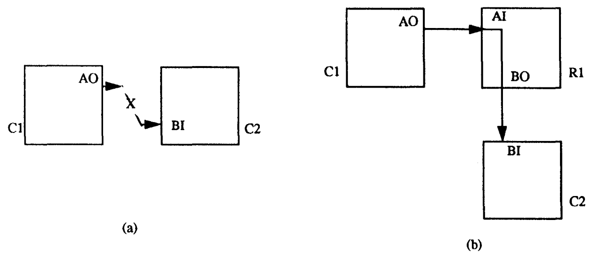

FIGURE 6 The scheme to get from pin AO of C1 to pin BI of cell C2.

egy. The placement tool cannot lend itself to considering both global issues and the detailed issues simultaneously as is possible in the flexible standard cell designs. In the coarse view of the problem, the placement tool must consider global connectivity and congestion estimates of the global routing regions irrespective of the detailed routing conventions. All constraints, including timing, can be viewed at a coarser level. Using the global solution as a guideline, the placement tool can next consider the local regions. Congestion at the detailed view can be determined by the demand and availability of routing resources in a local routing region. Detailed placement of macros must be achieved based on the local routing estimates while precisely satisfying all constraints at a detailed view. We thus use a hierarchical placement approach consisting of two phases: a global placement (or partitioning) phase followed by a detailed placement phase.

3.1. Phase 1: Global Placement

The objective of the global placement stage is to obtain a placement of the macros and pads with respect to global connectivity and congestion; and to satisfy all constraints at the global level. The blocks are the architectural boundaries of equal size sections of the chip each accommodating an $m \times m$ array of cells. The block boundaries provide a partitioning guideline for the global placer. The macros in the circuit are thus partitioned into the blocks during this stage. The ratio of the number of pads to the number of macros for these circuits is much larger than the corresponding ratio for standard cell designs. Each peripheral block caters to a set of pads and contains the only routing resources these pads have access to. For these reasons, the placement of the pads in this problem is critical. Pad placement must be performed with respect to direct I/O connectivity of the macros and thus must be performed simultaneously while macros are being partitioned. Long distance connections between blocks are made through long wires or buses in this architecture. For such straightforward bus connections, minimizing the total wire length effectively minimizes congestion in addition to reducing the global lengths of nets between blocks and I/Os. Thus, the routability of the *global* connections due to the chip I/O signals and the inter-block signals can be maximized by simply minimizing total wire length between blocks and pads. However, in the process of minimizing wire length, the circuit should be distributed throughout the chip as uniformly as possible. This is due to the following reasons:

1) The chip area is fixed and thus all of the chip core is available for use.
2) Dense packing of macros causes congestion and high demand for routing resources.

Thus the global placer must seek a partitioning in which each block is assigned a number of macros such that the total number of active cells in each block is comparable. The partitioning algorithm should also satisfy the timing constraints.

Recursive mincut bipartitioning strategies cannot be deployed successfully for global placement since issues such as total wire length and the length of critical signal paths cannot be controlled. Recursive mincut bipartitioning can only control the number of pinouts required by a partitioned block and this is inadequate. We have identified a need for an automatic N-way partitioner which understands the notion of distance between the various partitions. The partitioner should be able to control the number of pinouts on each partition and ensure that signal path lengths do not exceed their bounds while the total wire length is minimized.

3.1.1. Simulated Annealing based N*-way Partitioning*

The global placement and 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 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 penalties in [1]. Instead, during new state generation, the partitioner only picks moves which are feasible. This allows the algorithm to condense the search space of new states and thus improves run time. The new state generation function picks moves which involve both macros and I/O pads in order to accomplish pad placement at the same time as macro partitioning. The annealing schedule used is a statistically derived schedule proposed by Lam [10] and extended by Swartz [9].

Each block is considered a partition and the block edges are the cut lines for partitioning. During the partitioning process, an FPGA macro will move from block to block. Its location at any instant is taken to be the center of the block to which it currently belongs. A large number of partitions make the calculations of the wire length more precise, especially with respect to timing. However, with a large number of partitions, the search space for macro 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 setting the number of partitions equal to the number of blocks in this case. For chips with a lower number of blocks, the partitioner introduces additional cut lines to further divide each block into multiple partitions.

Cost Function C

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

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

Total Wire Length W

Since the final wire length of each net cannot be determined until the detailed placement and routing are done, the wire length of each net is estimated in the partitioning stage. The estimation uses the half perimeter of the minimum rectangle that encompasses the net. The total wire length estimate for a particular configuration of cells is the sum of three terms, W_{nrc}, the total wire length of the non-clock and non-reset nets (there are N_{nrc} such nets), W_c, the length of the routable portion of the clock nets (there are N_c such nets), and W_r, the length of the routable portion of the reset nets (there are N_r such nets).

$$W = W_{nrc} + W_c + W_r \tag{2}$$

$$W_{nrc} = \sum_{n=1}^{N_{nrc}} (S_x(n) + S_y(n)), \tag{3}$$

where $S_x(n)$ and $S_y(n)$ are the width and height of a minimum rectangle, respectively.

The wire length of the clock and reset nets are estimated slightly differently. The connection between the sinks of those nets and the feeding cell in the top or bottom edge of the core is implicit through the corresponding columns as shown in Figure 3. Hence, the y span from the source of the clock to the top edge of the core is the only part of the y span which should be included in the cost function. Similarly, the y span of a reset signal is only that from the source of the reset to the bottom edge of the core.

$$W_c = \sum_{n=1}^{N_c} (S_x(n) + (\text{chip\_top\_y} - \text{source\_y}\ (n))) \tag{4}$$

$$W_r = \sum_{n=1}^{N_r} (S_x(n) + (\text{source\_y}\ (\text{n}) - \text{chip\_bottom\_y})) \tag{5}$$

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 block structure used and thus is simpler and faster than previously reported methods [9,1]. Since detailed placement is going to follow this partitioning stage, the clusters of FPGA macros are assigned to the center of the blocks and the pins of a block are also taken to be at the center of the block.

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 lookup table. The x and y span of a net is calculated using the maximum and minimum of the active grid lines (*i.e.* these grids contain one or more pins) of the net.

Timing Penalty

The timing penalty in the cost function is calculated based on the slacks generated in the critical paths of the circuit during global placement. A critical path may consist of several nets. The timing penalty is minimized dynamically during placement. The total delay on a path p is the sum of the delay generated in the configurable logic blocks (CLB), $T_L(p)$, and the total interconnect delay, $T_R(p)$ [11].

$$T_{pd}(p) = T_L(p) + T_R(p) \tag{6}$$

$T_R(p)$ is the sum of all the constituent net routing delays, $T_R(n)$, due to the interconnections of the net.

$$T_R(p) = \sum_{\forall n \in p} T_R(n) \tag{7}$$

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

$$T_L(p) = N_D \cdot T_{CLB}, \tag{8}$$

T_{CLB} is the intrinsic delay of the CLB. 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.

Routing Delay: $T_R(n)$ is a function of the routing architecture of this FPGA, fanout of a connection, length of a connection, the process technology, and the programming technology. The two main components of $T_R(n)$ are delay due to the switches in the interconnect path and the parasitics of the wire segment. The total switching delay including the parasitics seen by the wire segments used by the net can be modeled as a lumped RC:

$$T_R(n) = R_{SW}C_{SW} = R_{SW}(C_g + C_p) \tag{9}$$

The drive resistance, R_{SW}, is the switching ON resistance. Since exact values R_{SW} are not easy to compute before detailed placement, we use pre-computed estimates of R_{SW} for this particular FPGA. C_{sw} is the switch loading capacitance which consists of the gate input capacitance, C_g, and the parasitic capacitance, C_p, of the wire segments used to form the interconnection. 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:

$$C_p = C_{L_h} S_x(n) + C_{L_v} S_y(n) \tag{10}$$

C_{L_v} and C_{L_h} are the capacitances (per unit length) of the vertical and horizontal tracks or buses in the routing architecture. Thus (9) can be expanded as:

$$T_R(n) = R_{SW}C_g + R_{SW}[C_{L_h} S_x(n) + C_{L_v} S_y(n)] \tag{11}$$

The total path delay for p is:

$$T_{pd}(p) = T_L(p) + \sum_{\forall n \in p} (R_{SW}C_g + R_{SW}[C_{L_h} S_x(n) + C_{L_v} S_y(n)]). \tag{12}$$

As reported in [9], 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 $T_{lb}(p)$ on the required arrival times. The penalty assigned for a path p is the amount the delay deviates from satisfying the bounds.

$$P(p) = \begin{cases} T_{pd}(p) - T_{ub}(p) & \text{if} \quad T_{pd}(p) > T_{ub}(p) \\ T_{lb}(p) - T_{pd}(p) & \text{if} \quad T_{pd}(p) < T_{lb}(p) \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

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

$$P_t = \sum_{p=1}^{N_p} P(p) \tag{14}$$

New State Selection Procedure

A key objective of the partitioner is to achieve roughly comparable utilization in each of the partitions. The utilization (U) in a partition as defined as the number of cells used by macros in that partition divided by the total number of cells in that partition. The *a priori* target or baseline utilization (U_B) for each partition is defined as the total number of cells used by macros throughout the chip divided by the number of cells on the chip. To obtain uniform 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 7. Here, U is the utilization of the partition before a move and U_{new} is the new utilization of the partition if the proposed move is accepted. The partitioner uses two main types of moves: a single object move or a pairwise interchange of two objects. For new state generation, we randomly select an object. If the object is a macro, a single macro move from one partition to another is attempted first. If such a move does not pass the utilization test, a pairwise interchange of two macros is attempted. If the object is a pad, a single pad move/pairwise interchange of two pads is attempted to another I/O slot depending on whether the second slot is empty or occupied, respectively.

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

FIGURE 7 Function for balancing utilization.

3.2. Phase 2: Detailed Placement

The global placer assigns a group of macros to a block. The objective of the detailed placement stage is to obtain an exact placement of the macros following the topological directions from the global placement results. The detailed placer must maximize the routability of the FPGA by considering factors which cause congestion at the detailed routing level. While maximizing routability, the detailed placer must precisely satisfy all of the constraints in order to generate a feasible placement.

In order to maximize routability for the placement, the amount of routing resources required for the placement must be minimized. Congestion in a local routing region can be determined by the demand and availability of routing resources for detailed routing. Since the total wire length model breaks down for estimating demands for routing resources at this level, the placer needs to pursue the factors which strongly affect routability for the given routing architecture. We studied the characteristics of the routing architecture and found that routability is strongly affected by factors such as: typical amounts of routing resources required for a macro configured in this architecture, the routing conventions typically used for various net topologies, flexibility of the macros in terms of their multiple versions or configurations. Without access to the gate level architectural details and direct knowledge of routing congestion from a detailed router, accurate estimation and prediction of exact routing demands is nontrivial at the placement level. However, we found that because there are limited routing resources (*e.g.*, buses) available for every cell to use in this architecture, increasing the number of potential routing cells around a macro and minimizing the utilization of routing buses are both important to maximize routability of the placement. Dense packing of macros causes congestion and high demand for routing resources. The chip area is fixed and all of the chip core is available for use. The macros must thus be placed such that they have suitable routing area available around them.

The detailed placer must also precisely satisfy all of the constraints in order to generate a feasible placement. The initial state of the detailed placement problem, obtained from the practitioner in the first phase, contains instances of macro overlapping. Overlapping must be removed. Timing violations and clock/reset conflicts must be eliminated. Our detailed placement algorithm is based on the following objectives:

1) eliminating overlapping;
2) eliminating clock/reset conflicts;
3) eliminating timing violations;
4) increasing the space between macros;
5) reducing the number of buses used.

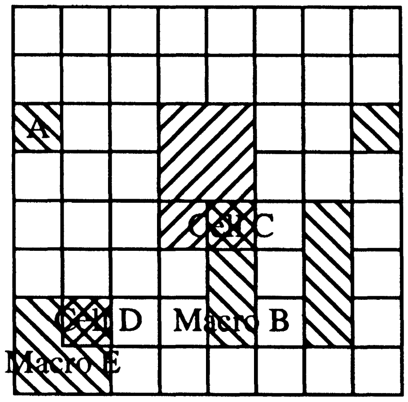

FIGURE 8 Examples of probe distance calculation.

Probe Distance

In order to measure the spacing between macros, we have introduced the concept of a *probe distance* (PD) from a macro. The probe distance from a cell of a macro in a given direction is the number of steps (cells) that the macro can move without hitting another macro or the boundary of the chip. In case another macro overlaps this macro, the probe distance will be a negative number with a magnitude equal to the number of steps needed to move the macro in the opposite direction to eliminate the overlap of the two macros. For example, in Figure 8, the probe distances from cell A towards the north, south, east, and west directions are 2, 3, 2, and 0, respectively. Since macros B and C overlap, the probe distance from cell C towards the north, east, and west directions are -1, -2, and -1, respectively.

Minimum Probe Distance

We then define the minimum probe distance towards a direction (say east) for a macro to be the smallest of the probe distances from all the cells on the east boundary. The minimum probe distances in the northeast, southeast, southwest, and northwest directions are defined in a similar manner. In this case the probes are made from the corresponding corner cells. For example, the minimum northeast (NE) probe distance from macro E is just the NE probe distance of cell D, or 1. The minimum probe distance (MPD) for the macro as a whole is defined as the smallest minimum probe distance among those in all eight directions. The MPD gives a measure of how close a macro is to the other macros. If the initial state of a placement problem contains instances of macro overlapping, the overlap of a macro can be reduced by increasing MPD.

Maximizing Routability

Given a particular macro, the more unused cells around its periphery, the more likely it is that sufficient routing resources exist for connections to all of the pins on the periphery of the macro. Furthermore, given a particular pin on a cell on the periphery of a macro, the larger the probe distance, the more likely it is that the routing to this pin can be successfully completed. A periphery cell which has a shorter probe distance is much more difficult to route to than a periphery cell which has a longer probe distance. To reflect this observation, we therefore developed the following inverse routability measure for a macro *m*.

$$\text{InverseRoutability} = IR(m) = \sum_{nc} \frac{1}{[PD(nc) + \varepsilon]^2} + \sum_{c} \frac{1}{[PD(c) + \varepsilon + 0.5]^2}, \qquad (15)$$

where the non-corner periphery cells of a macro m are represented by nc, and the corner periphery cells are represented by c. Note that when the expression in (15) is minimized, then the macro is positioned such that its $1/r^2$ repulsion forces to all of its neighboring macros is minimized. Our experiments have shown that $1/r^2$ repulsion forces are more effective than $1/r$ for this detailed placement problem.

The corner cells are treated differently for the following reason. The probe distance is zero for two cells adjacent at a corner as well as for two cells adjacent at a side, but the former is somewhat more routable, although not as much as two cells with a probe distance of one. Hence the extra factor of 0.5 is added to the probe distance for the corner cells. Since probe distances can be zero for non-overlapping macros, we add ε to the probe distance to assure numerical stability. Currently we use $\varepsilon = 0.5$. Figure 9.1 shows the probe distances for macro m. We compute its inverse routability measure as:

$$IR(m) = \frac{2}{1.5^2} + \frac{1}{2^2} + \frac{5}{2.5^2} + \frac{2}{3^2} + \frac{3}{3.5^2} + \frac{1}{4^2} = 2.468 \qquad (16)$$

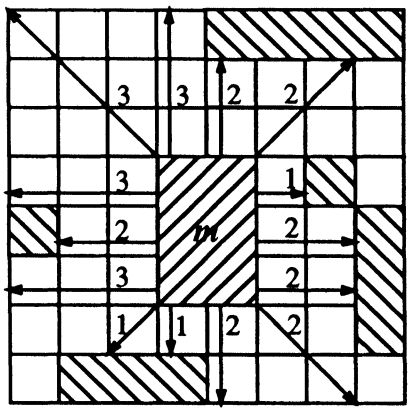

FIGURE 9.1 All of the probe distances for macro m.

Figure 9.2a illustrates the probe distances for macro m after it has been moved one step in the SE direction. Now,

$$IR(m) = \frac{2}{1^2} + \frac{5}{1.5^2} + \frac{1}{2^2} + \frac{3}{3.5^2} + \frac{2}{4.5^2} + \frac{1}{5^2} = 4.856 \qquad (17)$$

Note that this arrangement is much less routable as indicated by the sharp increase in the inverse routability measure. Finally, Figure 9.2b shows the probe distances for macro m after it has been moved one step in the NW direction.

$$IR(m) = \frac{1}{1.5^2} + \frac{1}{2^2} + \frac{7}{2.5^2} + \frac{2}{3^2} + \frac{1}{3.5^2} + \frac{1}{4^2} + \frac{1}{4.5^2} = 2.230 \qquad (18)$$

Clearly this arrangement is more routable, and this is reflected in a reduction of IR as well.

Bus Utilization

The buses traverse a length of a block in both horizontal and vertical directions. However, once a section of a bus in a block is used for a signal, the entire bus is dedicated to that signal and is not available for use by any other signal. Pins in the same column and same row can be connected using the same bus within the bounds of a block (Figure 10). The minimum number of buses required for a net traversing more than one block is equal to the number of blocks covered by this net in each direction. This number is given by the number of inter-block grid lines crossed by the net. Hence, we have formulated a measure for bus utilization by summing up the active grid lines (*i.e.* these grids contain one or more pins) of the net. To obtain the total bus utilization (*BU*), we sum the measure for all nets on all the macros.

Clock/Reset Conflicts

We use *NumClk [i]* to denote the number of different clock signals present in column *i*, and *NumSig [i] [j]*

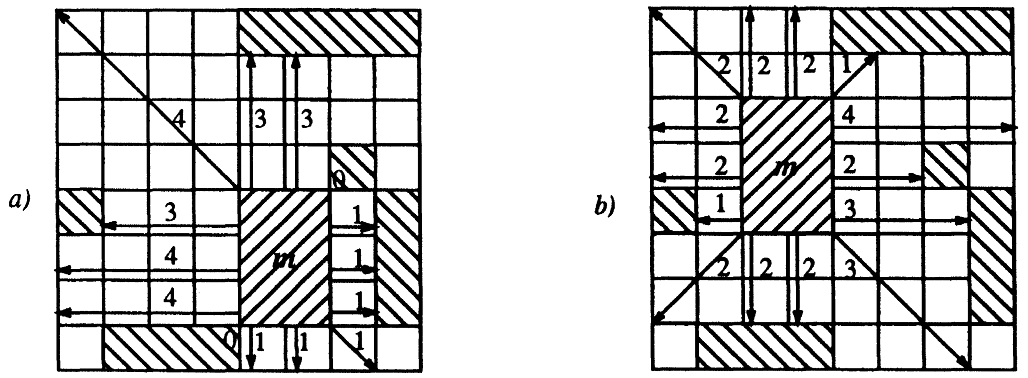

FIGURE 9.2 a) Probe distances for macro m after a one-step move in the SE direction. b) Probe distances for macro m after a one-step move in the NW direction.

to denote the number of pins which are located in column i and belong to clock net j. Since the architecture only allows one clock signal in every column, we define the number of clock conflicts as:

$$ClockConflicts = \sum_i \left((NumClk[i] - 1) \times \left(\sum_j NumSig[i][j] - MaxNumSig[i]\right)\right) \quad (19)$$

where,

$\sum_j NumSig[i][j]$ = total number of clock pins in column i,

$MaxNumSig[i]$ = maximum number of clock pins for any clock net in column i, and the leftmost summation in (19) is over all columns having $NumClk[i] > 1$.

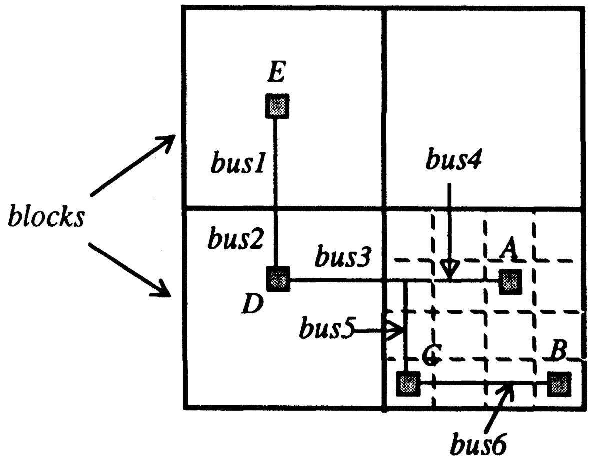

FIGURE 10 Inter-block and intra-block bus utilization.

Therefore, when there is only one clock signal in a column, we do not have conflicts in that column. When there is more than one clock signal present in the column, the net which has the most pins in the column is called the *dominating clock net*. All other nets are called *non-dominating clock nets*. The number of conflicts is equal to the number of pins of all non-dominating nets weighted by the number of clocks in this column (Equation (19)). Adding a dominating clock net pin to a column will not change the conflicts, but adding a non-dominating clock pin will cause the conflicts to increase, while adding another non-dominating clock net will increase the conflicts the most. A similar definition is used for reset conflicts. The sum of both clock and reset conflicts is denoted by *CRC*.

Detailed Placement Algorithm

The detailed placement algorithm is shown in Figure 11. The main idea is to randomly select a macro and move it one step (cell) in one randomly selected direction out of the eight possible directions. A move is accepted or rejected according to the *accept* function.

```
Detailed Placement Algorithm
1)  Randomly select a macro, m;
2)  Probe from m to get its MPD;
    If MPD(m) ≥ 0(no overlap with others),
        Calculate IR(m);
        Calculate CRC;
        Estimate routing resources (BU) used by the nets of m;
        Compute the new timing penalty, $P_t$;
3)  Randomly choose one of the eight directions;
4)  Move m one step in the selected direction;
5)  Do step 2 for the new location of m;
6)  Using the Accept function, determine whether to accept or reject
      this move;
7)  Continue until the number of continuously rejected moves exceeds a
      threshold.
```

```
Accept function
If MPD(m) < 0 (there is overlap), accept only when overlap is reduced or
   eliminated (i.e., MPD is increased);
If MPD(m) ≥ 0 (there is no overlap), accept when $P_t$ is not increased AND
   one of the following is true:
     1) IR(m) measure is reduced and CRC is not increased;
     2) CRC is reduced and MPD(m) is not decreased;
     3) BU is reduced and CRC is not increased.
```

FIGURE 11 Detailed placement algorithm.

The process continues until the number of continuously rejected moves exceeds a given threshold.

4. RESULTS

Table I contains a profile of the test circuits that we have obtained from an industrial source for this FPGA architecture. Most of these circuits have a combination of clocks and resets. Each circuit was placed using our placement approach, SOGFP, as well as the industrial placement tool from CLI. Each placement, whether obtained from CLI or SOGFP, was routed using the CLI router. In order to compare the quality of the placement tools, Table II shows the number of unrouted nets which remained after routing. In all cases, SOGFP outperforms the industrial

Table I FPGA circuit profile

Circuit	#Macros	#Nets	#Pins	#Clk/#Rst	#IO's	Avg. Fanout
p1	250	317	1060	0/0	60	3.34
p2	277	307	960	0/0	60	3.13
p3	439	385	1021	1/1	80	2.65
p4	359	459	1310	2/2	55	2.85
p5	329	316	1083	3/1	93	3.43
p6	212	275	819	1/1	69	2.98
p7	305	346	1047	2/2	60	3.03
p8	305	346	1047	2/2	60	3.03
p9	268	290	965	2/5	60	3.33
p10	314	345	1166	2/2	60	3.38
p11	314	345	1166	2/2	60	3.38
p12	314	345	1166	2/2	60	3.38

Table II Number of unrouted nets after using the same routing tool

Circuit	CLI	SOGFP	Reduction(%)
p1	3	0	100
p2	1	0	100
p3	1	0	100
p4	4	0	100
p5	0	0	—
p6	19	4	79
p7	23	4	91.3
p8	17	0	100
p9	10	0	100
p10	15	3	80
p11	18	4	77.8
p12	26	8	69.0
Average			90.6

tool. On average, SOGFP reduced the number of unroutes by 90%. Five circuits unroutable after CLI placement were 100% routable after using the placement obtained from SOGFP. Figure 12 shows an industrial FPGA circuit (p10) placed using SOGFP. The CPU time required for the largest circuit was about 30 minutes on a SUN SPARC 2.

Circuits with 10 or fewer unrouted nets can usually be fully routed by manual interaction. An improved router would likely be able to achieve 100% routability for those circuits. This is the subject of current research.

5. CONCLUSION

A new hierarchical placement approach has been developed for a sea-of-gates style FPGA architecture. In this two-phased approach, the circuit is physically partitioned into groups of FPGA macros over the available chip area and the pads are placed in the global placement phase. The global placement phase minimizes congestion estimates of the global routing regions and satisfies all constraints at a coarser level. Following the topological directions from the global placement phase, a detailed placement phase is executed to determine the exact location of each macro. The detailed placer maximizes the routability of the FPGA by considering factors which cause congestion at the detailed routing level and precisely satisfies all of the constraints. Despite having limited knowledge about the gate level architectural details, we have achieved a 90% improvement over the results produced by an industrial tool (the only tool available) developed specifically for this architecture.

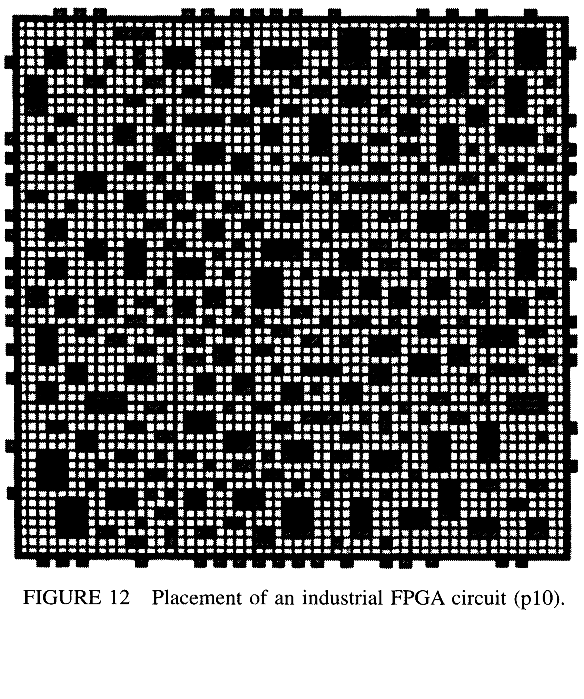
FIGURE 12 Placement of an industrial FPGA circuit (p10).

Acknowledgments

This research was funded by the Semiconductor Research Corporation, Intel Corporation, Digital Equipment Corporation, National Semiconductor Corporation, and Cadence Design Systems. The authors would like to thank NSC for providing the circuits.

References

[1] C. Sechen and K. W. Lee, "An improved simulated annealing algorithm for row-based placement," in *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1987, pp. 478–481.

[2] S. Nag and K. Roy, "Iterative wirability and performance improvement for FPGAs," in *Proc. 30th Design Automation Conf.*, 1993, pp. 321–325.

[3] S. Trimberger and M. R. Chene, "Placement-based partitioning for lookup-table-based FPGAs," *FPGA 92, First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992, pp. 136–142.

[4] N. Bhat and D. Hill, "Routable technology mapping for FPGAs," *FPGA 92, First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992, pp. 143–148.

[5] E. Walkup *et al.*, "Routing-directed placement for Triptych FPGA," *FPGA 92, First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992, pp. 33–38.

[6] J. F. Beetem, "Simultaneous placement and routing of the LABYRINTH reconfigurable logic array," in W. Moore and W. Luk, Editors, *FPGAs*, pp. 232–243, Abingdon EE&CS Books, Abingdon, England, 1991.

[7] S. Brown, R. J. Fancis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[8] B. Tseng, J. Rose, and S. Brown, "Using architectural and CAD interactions to improve FPGA routing architectures," *FPGA 92, First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992, pp. 3–8.

[9] 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.

[10] J. Lam and J. M. Delosme, "Performance of a new annealing schedule," in *Proc. 25th Design Automation Conf.*, 1988, pp. 306–311.

[11] 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.

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.

Bingzhong David Guan received BS degree from the University of Science and Technology of China, Heifei, China, in 1989, MA degree from the University of Southern California in 1990 and MSEE degree from the University of Washington in 1992. He is currently working towards the Ph.D. degree in electrical engineering at the University of Washington. His research interests include automatic layout of integrated circuits, particularly, FPGA and automatic standard cell generation to minimize chip area. He is a member of Eta Kappa Nu and IEEE.

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.