A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC

M. Slimane-Kadi, D. Brasen, G. Saucier
Institut National Polytechnique de Grenoble / CSI
46, avenue Félix Vialet, 38031 GRENOBLE cedex FRANCE
Tel: (33) 76-57-46-87, FAX: (33) 76-50-34-21, E-mail: saucier@imag.fr

FPGA Areas) Fast-Prototyping of System level design.
Field-Programmable Interconnect Chips and Devices (FPIC/FPID).

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) are rapidly replacing low end Gate-Arrays because of their fast turn around time and low cost. However, system level Printed Circuit Board (PCB) concerns such as package partitioning and inter-package routing remain as difficult as ever. In this paper, we present a fast prototyping system that uses inexpensive high-performance Field Programmable Interconnect Chips (FPIC). The FPIC form a fixed crossbar that is 100% routable when used in cooperation with automatic package partitioning and crossbar routing algorithms. Regular PCB performance can be obtained in less time to market and less cost.

1 INTRODUCTION

FPGA packages offer low cost and fast turn-around time solutions for small low volume circuits. However, PCB system level design becomes more of a problem. The small Size and IO

![Diagram of a proposed prototyping system for FPGA implementation.](image-url)

Figure 1: A Proposed Prototyping System For FPGA Implementation.
constraints of FPGAs cause the partitioning of circuits into greater numbers of packages. With more packages, more ordinary signal wires cross between the packages causing greater problems with PCB interconnect routing and interconnect delay. In many cases, the use of custom fabricated PCBs with traditional PCB design techniques can destroy all the cost and time advantages of FPGAs.

Prior work in FPGA PCB interconnect routing has focused on the fully automatic FPIC methods that provide reduced time to market on pre-fabricated PCBs [1][2][5]. The solution of [2] has a dynamic interconnection network of an ASIC crossbar with 936 usable IOs that can connect directly any of the IO pins. For example, 7 Xilinx XC4010 FPGAs with 128 IO pins can be directly connected with one crossbar chip. A major problem is that the ASIC crossbar used is expensive, reducing some of the price advantages associated with FPGAs.

The solution of [2][5] is for high scale prototyping of complex architectures. Each PCB allows for a circuit to be implemented onto 14 Xilinx XC3090 FPGAs which are each connected by programming the interconnect matrix of 32 Xilinx XC2018 FPGAs. A second crossbar network on board of 64 Xilinx XC2018 FPGAs permits the cascading with 7 other PCBs allowing for an 8 card system. This system gives improved flexibility and on-board circuit density. However, the interconnection network has limits on the number of connections between FPGA chip pairs. Also the use of RAM inter-package connections within Xilinx chips cause a serious reduction in the performance of the entire circuit.

A new FPIC network has been designed for low-cost high performance designs. The solution allows for fast prototypes much like the Quickturn with 8 Xilinx XC4010 chips for implementation of circuits. The difference is the use of 16 Actel A1280 chips for higher interconnect speeds than those of the Xilinx crossbars used by Quickturn. Using standard Actel and Xilinx chips provides a cheaper and more flexible solution than the Aptix ASIC crossbar. Like the Quickturn solution, this fast-prototyping system is theoretically not always routable. However, the use of a specialized FPGA package partitioning algorithm in combination with a predictable crossbar routing algorithm allow for 100% routability. These two software packages, both presented here, provide quick and accurate mapping of the circuit to the FPGAs for fast-prototyping.

This paper first describes the inexpensive fast-prototyping FPGA system which uses an FPIC crossbar designed for high performance. Crossbar routing configurations and partitioning
constraints are listed. The modified FPGA package partitioning algorithm of [4] allows for
100% routing of the fixed crossbar. The partitioned circuits are then connected by a crossbar
routing algorithm that provides predictable results. Solutions to this routability problem are then
discussed in detail. Experimental results show that the partitioning/routing algorithms presented
here result in routable PCB prototyping boards at lower cost and high performance.

2 AN INEXPENSIVE FAST-PROTOTYPING SYSTEM

The fast-prototyping FPIC proposed in Figure 6 has 8 Xilinx XC4010 chips for
implementation of circuits much like the Quickturn or Aptix solutions. The difference is the use
of 16 Actel A1280 chips for high performance FPIC crossbars at lower cost and greater
flexibility. Functional frequencies of the clock can be set anywhere from 1MHZ to 9MHZ.

![Diagram](image)

**Figure 2: General Scheme For An Improved Performance FPIC Network.**

As with all fast-prototyping systems, a prefabricated PCB board is provided with all
Xilinx/Actel connections fixed. During prototyping, the FPIC crossbars are programmed to
make the desired inter-chip connections of the circuit to be relized. The Aptix ASIC and
Quickturn Xilinx crossbar chips then have the ability to be reprogrammed. It is relized here that
during the circuit prototyping process, crossbar connections do not need to be changed. The
Actel crossbar with its anti-fuse technology is programmed once, but provides a cheaper
solution with high-performance interconnects.

The Xilinx4010/Actel1280 parts chosen have matching pin connections for optimal routing of
general circuits with the least chips. Each Xilinx chip is connected to each other Xilinx chip
through a matrix of 16 Actel chips. There are 16 buses of 8 bits per Xilinx chip. Each of the
buses is realized by an Actel chip. There are 29 primary inputs/outputs per Xilinx chip (232
primary inputs/outputs total). They serve to send and receive signals, from or to the exterior, which is necessary for proper functioning of the logic implanted in the Xilinx FPGAs.

3 CROSSBAR ROUTING FOR FPIC CONFIGURATIONS

A study of the occupation of Actel A1280 FPGAs has been performed for several types of routing in the crossbar; "bit by bit", "by 2 bits", "by 4 bits", or "by 8 bits". Only the first realizes a true crossbar. The "last by 8 bit" network has the largest constraints on routing while also requiring the least routing resources. This is demonstrated by using the number of logic modules occupied by different multiplexors types as shown in Table 1.

<table>
<thead>
<tr>
<th>Mux Types</th>
<th>16:1 Mux</th>
<th>8:1 Mux</th>
<th>4:1 Mux</th>
<th>2:1 Mux</th>
</tr>
</thead>
<tbody>
<tr>
<td># Modules</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Modules Required To Realize Different Mux Connection Types.

The results of this study are summarized in Table 2. For the "bit by bit" approach, the size of the crossbar is too big to be realized on a network of Actel A1280 chips. For the "by 2 bit" approach, the crossbar occupies 75% of the routing pin resources of the A1280. However, the resources are not sufficient when considering the control part of the system. The routing utilization of the "by 4 bit" solution imposes constraints on the partitioning of the prototyping architectures that makes automatic FPGA package partitioning difficult.

<table>
<thead>
<tr>
<th>Route Bit By Bit</th>
<th>Route By 2 Bits</th>
<th>Route By 4 Bits</th>
<th>Route By 8 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 16:1 Muxs</td>
<td>160 8:1 Muxs</td>
<td>80 16:1 Muxs</td>
<td>80 8:1 Muxs</td>
</tr>
<tr>
<td>160 8:1 Muxs</td>
<td>160 4:1 Muxs</td>
<td>160 2:1 Muxs</td>
<td>80 2:1 Muxs</td>
</tr>
<tr>
<td>170%</td>
<td>78%</td>
<td>45%</td>
<td>26%</td>
</tr>
</tbody>
</table>

Table 2: Percent A1280 Utilization And Muxs Available For Different Routing Schemes.

4 PARTITIONING CONSTRAINTS

If there are a sufficient number of ACTEL crossbars, the system is fully routable since there is no routing blockage. However, there are certain cases where all the demands of routing can not satisfy. One has to search to express the limits of this solution by the constraints of the network (i.e., maximum number of connections possible between Xilinx chips). A first constraint is that each Xilinx chip with 128 IO pins does not have more than 128 connections,

$$\forall j \in [1..8], \sum_{i \neq j} N_{ij} < 128$$

where $N_{ij}$ represents the number of connections between the Xilinx chip $i$ and the Xilinx chip $j$. This first constraint is often not sufficient for resolving the blockage of the network. Suppose
that the architecture to prototype is partitioned into three Xilinx chips X1, X2, and X3, and that the routing constraints between these three Xilinx chips are those shown in Figure 5.

![Figure 3: Example Of Constrained Crossbar Routing Between 3 Xilinx Chips](image)

One can not satisfy all the routing demands as shown in Figure 6. If one satisfies the demands of the Xilinx chip X1, the available ports (i.e., 8 bit pin sets) are 1 through 8 on Xilinx chip X2. X1 and X2 are thus connected with crossbars 1 through 8. The ports used to connect X1 with Xilinx chip X3 are ports 9 through 16. Connections between X1 and X3 are thus realized with crossbars 9 through 16. One can not fully connect all of the 32 ports between the Xilinx chips X2 and X3 since this would require an additional 8 crossbars.

![Figure 4: Crossbar Routing Limitations Of Constrained Example.](image)

This blockage to full routability can be expressed by the following inequality,

\[
\forall i \in [1 \ldots 8], \forall j \in [1 \ldots 8] \sum_{L} (8 - \sum_{k \neq j} R_{ik}^{L}) \geq N_{ij}, R_{ij} \leq 8
\]

where \( R_{ik}^{L} \) is the number of connections to the interior of crossbar \( L \) between port \( i \) and port \( j \), and \( N_{ij} \) is the number of connections between Xilinx chip \( i \) and \( j \). A connection request between Xilinx chip \( i \) and \( j \), placed on any random crossbar, moves the port \( i \) and the port \( j \) of that
crossbar. Thus \( \sum_{k \neq j} R_{ik}^L \) indicates the connections already taken from port \( i \) towards any other random port on the crossbar \( L \), except the port \( j \). The term \( \sum_{k \neq j} R_{ik}^L \) corresponds to the number of pins remaining available at the end of port \( i \), and that can be realized by port \( j \). The formula expresses the number of connections possible to leave each port \( i \) towards the port \( j \) and the interior of each crossbar. This number should be superior to \( Nij \). It must not saturate a crossbar by reporting another. It must be tried again to meet routing requests on the system of crossbars rather than block a crossbar with a request.

5 FPGA PACKAGE PARTITIONING WITH CROSSBAR CONSTRAINTS

In this section, a modified cone partitioning algorithm of [4] is presented. Cone partitioning clusters highly connected groups of nodes based on combinational cone structures. Cone structures are minimum-cut structures because they outline regions of low fan-out. A fanout greater than one results in overlapping sections between cones. The partitioning is performed by finding regions of cone overlap created by net fanout. The modification allows for the \( Nij \) constraint on the number connections between of cluster pairs that ensures 100% routability.

Modified Cone Partitioning Algorithm:

1) Create a \( CRoot \) list from combinational node children (i.e., inputs) of all primary output nodes and sequential element nodes. This list size is the number of cones in the circuit. Note for combinational circuits there can never be more cones than primary outputs, but there can be more primary outputs than cones. Since sequential elements can have multiple inputs, the same rule does not apply for sequential circuits.

2) Scan combinational nodes (of the circuit) from each node in the \( CRoot \) list to primary inputs or sequential elements. During each scan, add to each combinational node a label corresponding to the entry number of the cone root in the \( CRoot \) list. Each combinational node has a resulting label list \( (LL) \),

\[
LL(Nd) = \bigcup_{i=1}^{\#P(Nd)} LL(P(Nd)_i)
\]

where \( \#P(Nd) \) is the number of parents \( P(Nd)_i \) of combinational node \( Nd \). If there does not exist a parent with an equal label list, then add combinational node \( Nd \) to the \( CRoot \) list since it is a cluster root with fan-out between cones. Note that these new \( CRoot \) entries have label list sizes greater than 1. The new \( CRoot \) list size is the number of clusters in the circuit.

3) Cluster overlapping cone sections by grouping combinational nodes with common label lists. Each cluster set has unique combinational nodes define by

\[
Cluster_i = CRoot_i \cup \left\{ Nd \mid \exists P(Nd), \left[ LL(Nd) = LL(P(Nd)_i) \right] \land \left( LL(Nd) = LL(CRoot_i) \right) \right\}
\]

where \( CRoot_i \) is the cluster root node of \( Cluster_i \) and \( P(Nd)_i = CRoot_i \) is possible. If the condition holds

\[
\forall i \neq j (LL(CRoot_i) \cap LL(CRoot_j) = \emptyset)
\]
then $\#Clusters = \#Cones$ and no cones overlap. Circuits having many more clusters than cones indicate high cone overlap and high fanout.

4) Add sequential elements to clusters for reduced Size/IO ratio. This means adding to clusters with the largest number of common interconnections.

6) Merge clusters containing nodes of critical paths.

5) Merge each cluster pair $(i,j)$ if the resulting cluster $M$,
   - is smaller than Xilinx package constraints
     \[ \text{Size}_M < \text{MaxSize}_{pk}, \quad \text{IO}_M < \text{MaxIO}_{pk} \]
   - has less than maximum pair-wise chip connections
     \[ \forall k \in \text{M}(\text{IO}_M \leq N_{km}) \]
   - minimizes the number of packages which for FPGAs means maximize the cost ratio
     \[ \text{Cost}_M = \frac{\text{Size}_M}{\text{IO}_M = \text{IO}_i + \text{IO}_j - (2 \times \text{IO}_{common})} \]
   where $\text{IO}_{km}$ is the number of connections between cluster $k$ and cluster $M$ and $N_{km}$ the maximum number of cluster pair connections for which the fixed the Actel crossbar was designed. This constraint ensures 100% crossbar routability.
   - minimizes the number of packages which for FPGAs means maximize the cost ratio

When cluster merging is finished, the remaining clusters are considered as final package partitions. Each resulting IO pin of each partition will become a package pin connected via a printed circuit board (i.e., PCB). To drive the loads associated with PCB interconnections, input/output buffers must be added to the IO pins of each partition.

6 CROSSBAR ROUTING ALGORITHMS

Two crossbar routing algorithms have been tested with various pin constraints. These constraints indicate the number of pins required to connect 2 Xilinx chips. We have tried a total of 4750 examples of which 250 use 10% of the routing resources, 250 use 15%, 250 use 20%, ..., 250 use 95%, and a final 250 use 100%. An example of x% resource utilization signifies that the constraints of the example use x% of the 1024 total pins in the network.

Both algorithms produce predictable results. The first algorithm searches for several crossbars at a time to entirely satisfy a constraint. The crossbar is kept in memory and made available to each constraint. One starts by routing the most pins possible on the crossbars who have the most pins routed.

FPIC Routing Algorithm 1:
1) Read a constraint such that $\text{pins} = function\_pin(constraint)$. 
2) Set crossbar = 0, route = function_route(constraint), table[crossbar]=route.
3) If route = pins, then the constraint is satisfied and go to step 1. Otherwise the constraint is not met and further searching is required.
4) If crossbar = 15, then set rem = pins. Otherwise increment crossbar and go to step 3.
5) If max(table) and rem are not 0 set; crossbar=max_index(table), rem=rem-table[crossbar], table[crossbar] = 0, and go to step 5.
6) If rem = 0 then the constraint is satisfied and go to step 1.
7) If max(table) = 0 and the constraint is still not entirely met, produce an unroute+ go to step 1.

where the size of table is a the number of crossbars and serves to put in memory the available pins on each crossbar of each constraint. Calculated by function_pin, pins is a variable that contains the number of constrained pins to route. Rem is a variable that contains the number of pins remaining to be routed to meet the constraint. Max is a function who calculates the maximum available pins such that max=0 implies that there are no more available pins on any crossbars. Max_index is a function that indicates the crossbar on which the available pins are maximum.

The second algorithm is close to the first. The main difference is that there has been some preprocessing of the constraints. The constraints on the number of connections between chip are sorted in decreasing ordered. The is an improvement because the first connections of the network routed are the most difficult to route.

7 EXPERIMENTAL RESULTS

In this section, experimental results are given for crossbar routing Algorithms 1 and 2, which have been programmed in C language. Experiments with partitioning algorithm, also programmed in C, were performed on industrial testcases and always resulted in partitioned Xilinx chips that were 100% routable. The maximum Xilinx chip pair IO constraint that permits this partitioning result is due to the predictable nature of the routing algorithms.

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>% Occ</th>
<th>Pins To Route</th>
<th>Pins Routed</th>
<th>% Succ</th>
<th>% Occ</th>
<th>Pins To Route</th>
<th>Pins Routed</th>
<th>% Succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>10</td>
<td>26622</td>
<td>26622</td>
<td>100.00</td>
<td>9.</td>
<td>65</td>
<td>167726</td>
<td>167336</td>
</tr>
<tr>
<td>2.</td>
<td>20</td>
<td>52116</td>
<td>52116</td>
<td>100.00</td>
<td>10.</td>
<td>70</td>
<td>180112</td>
<td>179352</td>
</tr>
<tr>
<td>3.</td>
<td>30</td>
<td>78098</td>
<td>78098</td>
<td>100.00</td>
<td>11.</td>
<td>75</td>
<td>193032</td>
<td>191344</td>
</tr>
<tr>
<td>4.</td>
<td>40</td>
<td>103742</td>
<td>103742</td>
<td>100.00</td>
<td>12.</td>
<td>80</td>
<td>206072</td>
<td>202282</td>
</tr>
<tr>
<td>5.</td>
<td>45</td>
<td>116100</td>
<td>116100</td>
<td>100.00</td>
<td>13.</td>
<td>85</td>
<td>218634</td>
<td>212396</td>
</tr>
<tr>
<td>6.</td>
<td>50</td>
<td>129130</td>
<td>129130</td>
<td>99.98</td>
<td>14.</td>
<td>90</td>
<td>231466</td>
<td>221320</td>
</tr>
<tr>
<td>7.</td>
<td>55</td>
<td>142246</td>
<td>142166</td>
<td>99.84</td>
<td>15.</td>
<td>95</td>
<td>243996</td>
<td>230672</td>
</tr>
<tr>
<td>8.</td>
<td>60</td>
<td>154708</td>
<td>154538</td>
<td>99.40</td>
<td>16.</td>
<td>100</td>
<td>252818</td>
<td>235630</td>
</tr>
</tbody>
</table>

Table 3: Results Of Algorithm 1.

The results of algorithm 1 are presented in the Table 3. The Table lists the percent theoretical pin occupation (% Occ), number of pins that need to be routed (Pins To Route), number of pins
that were routed successfully (Pins Routed), and the percentage Cost of successful routings (% Succ). The cost of successfully routing is Cost = (number of pins routed * 100)/(number of pins to route). Theoretical pin occupation cost corresponds to the percentage of network resource utilization that is an example of x%. These initial results represent a reasonable routing since many constraints can be routed (i.e., unroutes are not produced until 50% theoretical pin occupation has been reached).

<table>
<thead>
<tr>
<th>% Occ</th>
<th>Pins To Route</th>
<th>Pins Routed</th>
<th>% Succ</th>
<th>% Occ</th>
<th>Pins To Route</th>
<th>Pins Routed</th>
<th>% Succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>26622</td>
<td>100.00</td>
<td>9</td>
<td>65</td>
<td>167726</td>
<td>99.77</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>52116</td>
<td>100.00</td>
<td>10</td>
<td>70</td>
<td>180112</td>
<td>99.58</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>78098</td>
<td>100.00</td>
<td>11</td>
<td>75</td>
<td>193032</td>
<td>99.13</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>103742</td>
<td>100.00</td>
<td>12</td>
<td>80</td>
<td>206072</td>
<td>98.16</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>116100</td>
<td>100.00</td>
<td>13</td>
<td>85</td>
<td>218634</td>
<td>97.15</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>129130</td>
<td>100.00</td>
<td>14</td>
<td>90</td>
<td>231466</td>
<td>95.62</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>142246</td>
<td>100.00</td>
<td>15</td>
<td>95</td>
<td>243996</td>
<td>94.54</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>154708</td>
<td>100.00</td>
<td>16</td>
<td>100</td>
<td>252818</td>
<td>93.20</td>
</tr>
</tbody>
</table>

Table 4: Results Of Algorithm 2.

The results of algorithm 2 are presented in Table 4. The results of algorithm 2 are better than those of algorithm 1 as shown in Figure 5. By sorting the constraints in decreasing order, the hard to route large nets can be routed first. The smaller nets routed last are more easily routed through the remaining pins.
CONCLUSION

A new fast-prototyping system has been presented. This inexpensive high-performance system is implemented using Xilinx logic and Actel interconnections. Since the crossbar interconnections are fixed after programming, the system relies on FPGA package partitioning and crossbar routing algorithms to ensure 100% routability. The FPGA package partitioning algorithm presented in [4] has been modified to meet maximum IO connection constraints between Xilinx chip pairs specifiable prior to using the crossbar routing algorithms listed. As experimental results demonstrated, these specifiable constraints are possible due to the predictable nature of the crossbar routing algorithms.

REFERENCES