BORG: A Reconfigurable Prototyping Board using Field-Programmable Gate Arrays

Pak K. Chan*, Martine D.F. Schlag†, and Marcelo Martin‡§
Computer Engineering
University of California, Santa Cruz
Santa Cruz, California 95064

Abstract

Field-Programmable Gate Arrays (FPGA) provide a medium to accelerate the process of prototyping digital designs. For designs with multiple FPGAs that need to be connected together, the bottleneck is now the process of wire-wrapping, bread-boarding, the construction of a printed circuit board, or the construction of a multi-chip module, which cannot be carried out until all FPGA designs are routed. It is because locking or preassigning I/O blocks often prevent FPGA placement/routers from completing the routing.

We exploit the reprogrammability of FPGAs and use them for routing. To experiment with the idea, we constructed a PC-based prototyping board that contains two "user" FPGAs, two routing FPGAs, and a fifth FPGA that implements the glue logic to the PC bus. To facilitate the design process using the new prototyping board, we developed algorithms and tools that automatically configure the routing FPGAs. We describe the options that we have examined during the development of this board, and how we arrive at some design decisions.

The toolset, user FPGAs, and the routing FPGAs and the reprogrammability of the FPGAs serve to further reduce the time/cost of constructing prototypes using FPGAs.

*Supported in part by NSF Grant MIP-9111607
†Supported in part by NSF Presidential Young Investigator Grant MIP-8896276
‡Supported by NSF REU Supplement 1991
§The authors are grateful to the support and cooperation of Xilinx, Inc.

1 Programmable interconnect

Prototyping is a critical phase of the design of a digital system. The traditional techniques of connecting the parts in the prototype are wire-wrapping and bread-boarding, which are often time consuming and error-prone.

To alleviate this problem, some prototyping boards have PC-interface logic built-in to the FPGA prototyping boards [1, 2]. Connections between parts on the prototyping boards still require manual wiring.

We desire a prototyping board with electronically programmable interconnect between the parts. The board has the benefits of being easily alterable, reprogrammable, automatic, reusable, and correct.

Consider a prototype that requires two user FPGAs 'X1' and 'X2', as shown in Figure 1.

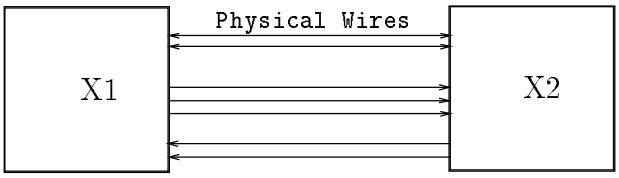

Figure 1: Two FPGAs need connections using external wires

One way of introducing programmable interconnect between 'X1' and 'X2' is having a programmable chip 'R' with double the amount of I/Os available inserted between 'X1' and 'X2'. This is depicted in Figure 2. For example, if 'X1' and 'X2' each has 84 pins, then 'C' has to have at least 168 pins (actually, the next available standard package is 175 pins). This scheme doesn't really solve the problem, there are two arguments against it. First, why not

simply use the 168-pin package to accommodate the logic of 'X1' and 'X2'? Second, it won't work if each of 'X1' and 'X2' has 168 pins.

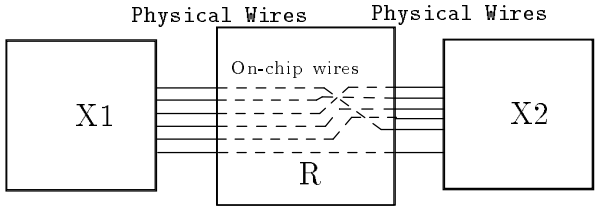

Figure 2: Two FPGAs need connections using on-chip wires, using one big chip for interconnect

We examine the idea of using two routing chips 'R1' and 'R2' to serve as interconnects between 'X1' and 'X2', as shown in Figure 3. In this case, all chips have the same number of I/O pads. The main issue is to determine the physical wiring pattern among the chips. We consider two possibilities:

By faces: The pins on the west face of 'X1' are physically connected to corresponding pins on the east face of 'R1', and so forth, forming a torus of connections among the faces.

Alternating: The even number pins of 'X1' are physically connected to the even number pins of 'R1', the odd number pins of 'X1' are connected to the odd pads of 'R2'.

So far, we have simplified the problem somewhat. In fact, the routing chip 'R1' needs to spend some pins to interact with the PC, and a static RAM is attached to 'R2'. This precludes the full connectivity between 'X1' and 'X2'. This issue will be addressed in the next section.

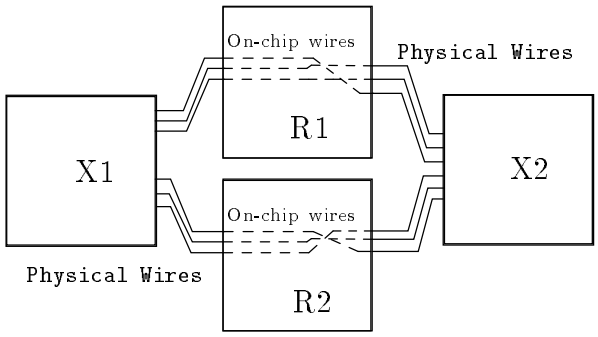

Figure 3: FPGAs need connections using on-chip wires, using two chips for interconnect

2 Algorithms for pad re-assignment

In the BORG design process, the final step of wire-wrapping or bread-boarding the connections between the user FPGAs, the PC interface and peripheral devices, is replaced by the task of configuring the two routing FPGAs to achieve these interconnections.

Unfortunately, since the pins of the FPGAs are hardwired on the board, some of the pad assignments produced by the place and route tools may need to be altered. For the routing FPGAs to achieve the required interconnects,

1. each net assigned to pads on both user FPGAs must be assigned to pads wired to the same routing chip, and
2. each net between the user FPGAs and the PC or a peripheral device, must be assigned to a pad wired to the appropriate routing FPGA for the host or peripheral device connection.

The nets must first be assigned to pads of the user FPGAs in a manner consistent with these two conditions, for it to be possible to configure the two routing FPGAs to achieve the required interconnect. Hence it might be necessary to reassign the user FPGA pads to meet conditions 1 and 2. This reassignment of pads on the two user FPGAs must be handled carefully as relocating nets can easily make a dense design unroutable.

Pad assignment (or pin assignment) can be represented as a matching problem on a bipartite graph where the nets and pads are the vertices, and there is an edge between a pad and a net if the net can be assigned to that pad, as shown in Figure 4. A maximum matching for this graph provides a pad assignment for each net, if there is one. When more than one maximum matching is possible, weights can be assigned to the edges in such a way that finding a maximum weight matching yields a pad reassignment whose total cost is minimized. This problem (also know as the linear assignment problem) is well understood and can be solved both by linear programming and specialized algorithms [3, 4].

Assigning nets to pads of the two user FPGAs in order to satisfy conditions 1 and 2 above involves solving two such bipartite matching

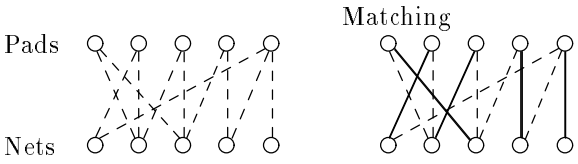

Figure 4: General pad assignment problem and matching on a bipartite graph

problems, one on each user FPGA with the additional constraint:

> If all pads wired to 'R1' are of color 1 and those wired to 'R2' are of color 2, then each net must be assigned to pads of the same color.

Nets which must be assigned to a pad wired to a specific routing FPGA can be easily handled in this framework by merely deleting edges from the net's vertex to pads of the wrong color. We call these nets *forced nets*.

Unfortunately, the algorithms for the bipartite matching problem do not readily extend to this problem. The basic technique of searching for an augmenting path fails since moving a net to a pad of a different color may also require moving the net to a pad of the other color on the other user FPGA. Hence augmenting a matching, in this case, consists of finding augmenting subgraphs in both bipartite graphs in which neither or both pads of each net switch color except for the nets which originally did not satisfy condition 1 or 2, see Figure 5.

The bipartite matching problem can also be solved using network flow techniques. However, adapting this view to solve our problem yields a two-commodity flow problem. In general graphs, the two commodity flow problem is NP-complete. Whether or not there is a polynomial time to solve it for the shallow graphs corresponding to our matching problem is open. Our approach has been to look for a limited type of augmenting subgraphs to improve the current matching. The types of subgraphs considered are:

Sliding path: this is an augmenting path consisting of pads all of the same color and ending with a currently unassigned pad. Figure 6 illustrates the pad reassignment using this technique. The arrows indicate the direction of reassignment.

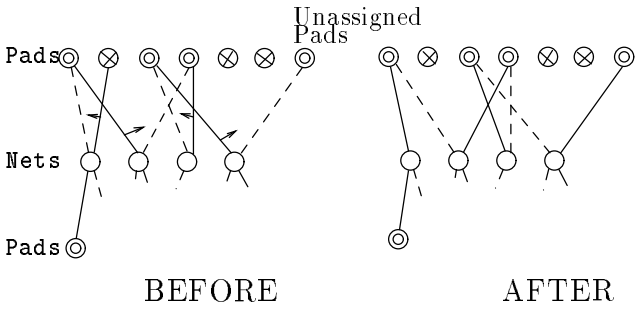

Figure 6: Sliding path

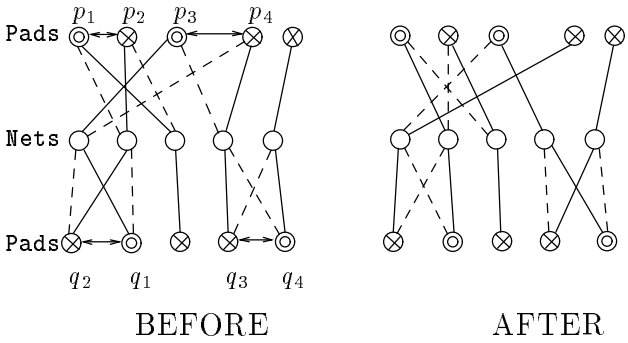

Figure 7: Ping-pong subgraph

Ping-Pong subgraph: this is an augmenting subgraph formed by a sequence of pairs of pads alternating between the two user FPGAs which can exchange nets. Specifically, a sequence of distinct pads

$$p_1, p_2, q_1, q_2, p_3, p_4, q_3, q_4, \cdots$$

such that for each i, 1) p_{2i-1} and p_{2i} are pads of the opposite color which can exchange net assignments, 2) q_{2i-1} and q_{2i} are pads of the opposite color which can exchange net assignments, 3) p_{2i} and q_{2i-1} are assigned to the same net, and 4) q_{2i} and p_{2i+1} are assigned to the same net. The path must terminate either with an unassigned pad, a pad whose net is currently assigned to two pads of opposite color, or a sequence of pads forming a sliding path of the right color. Figure 7 illustrates a pad reassignment using this technique. The double arrows indicate the swap.

Using only ping-pong subgraphs and providing each pad with a neighbor of the opposite color to swap nets with, guarantees the existence of a legal net assignment when there are no forced

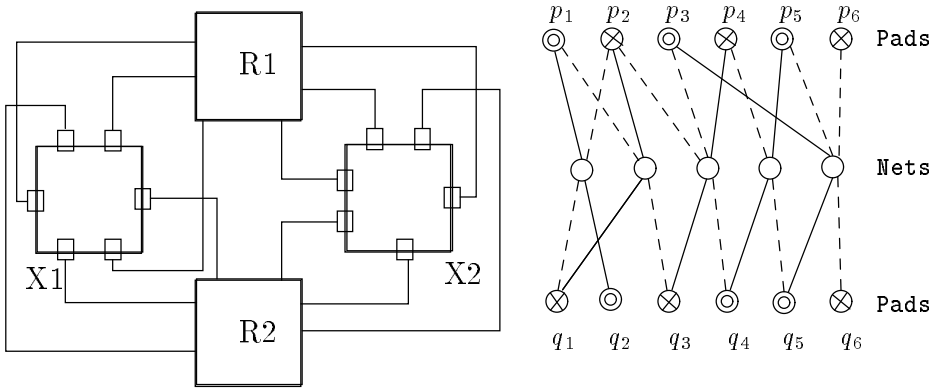

Figure 5: Pad reassignment problem as dual matching on bipartite graphs

nets. This observation supports the use of the alternating wiring board as described below.

Reassigning the pads on a densely populated FPGA may make it difficult to re-route. To minimize the movement of pads, our algorithm starts with the initial pad assignment and attempts to augment it progressively by allowing swaps of increasing distance. This is controlled by constructing the bipartite matching graphs based on a swap-window parameter. The parameter can be set and increased independently on the two user FPGAs. This can be used to limit pad movement on one FPGA more than the other if it will be more difficult to re-route than the other. Once an assignment satisfying conditions 1 and 2 has been found, the pad assignment can be improved by performing a weighted bipartite matching among all pads of the same color.

We experimented with two different board wirings and three designs. The two board wiring configurations we considered were:

Face aligned pads: The north and south faces of the user FPGAs are assigned to the north and south faces of the two routing FPGAs, and etc, forming a torus.

Alternating pads: On each user FPGA the usable pads alternate color.

Note that each of these board wiring configurations actually provide the symmetric configuration as well since the the two user FPGAs are interchangeable. We tried three designs which had the characteristics illustrated in Table 1.

The designs using XC3020pc84 packages did not require any placement iterations; the original placements were simply rerouted using the new pad assignments. The XC3042pc84 design required both placement and routing iterations. This design was difficult to place and route even when the pad assignment was not fixed.

3 Features of BORG

We highlight some features available in the new prototyping board:

1. 7-segment displays, LEDs, push buttons
2. stand-alone mode with serial download
3. standard 8-bit PC bus I/O interface
4. 25-pin external connectors (not available in the wire-wrapped prototype as shown in Figure 8)
5. small sea of holes proto-area for additional parts (not available in the wire-wrapped prototype as shown in Figure 8)
6. dual-ported 8K static RAM

A wire-wrapped prototype of the prototyping board is shown in Figure 8. A 84-pin FPGA XC3020 'X0' serves as the "glue" logic between the PC. It is also responsible for supplying the bit streams to configure the other four FPGAs using the peripheral mode [5].

Acknowledgements

This work is supported in part by the National Science Foundation. The authors are grateful to Xilinx Inc., DATA I/O Corporation, OrCAD, and ALDEC for their generous support and donations. We are also indebted to Bill Stevens for his technical support throughout the project.

	package	CLBs	IOBs	forced nets	swap-window alt.	swap-window face	place & routing iterations alt.	place & routing iterations face
amU1	XC3020pc84	43	27	6	1	6	0,1	0,1
amU2	XC3020pc84	62	21	0	1	6	0,2	0,1
rjU1	XC3020pc84	41	30	10	1	6	0,1	0,1
rjU2	XC3020pc84	48	20	0	1	6	0,1	0,2
maU1	XC3020pc84	58	40	9	4	8	0,1	0,1
maU2	XC3042pc84	116	52	21	4	8	5,1	6,2

Table 1: Results of two different wiring configurations

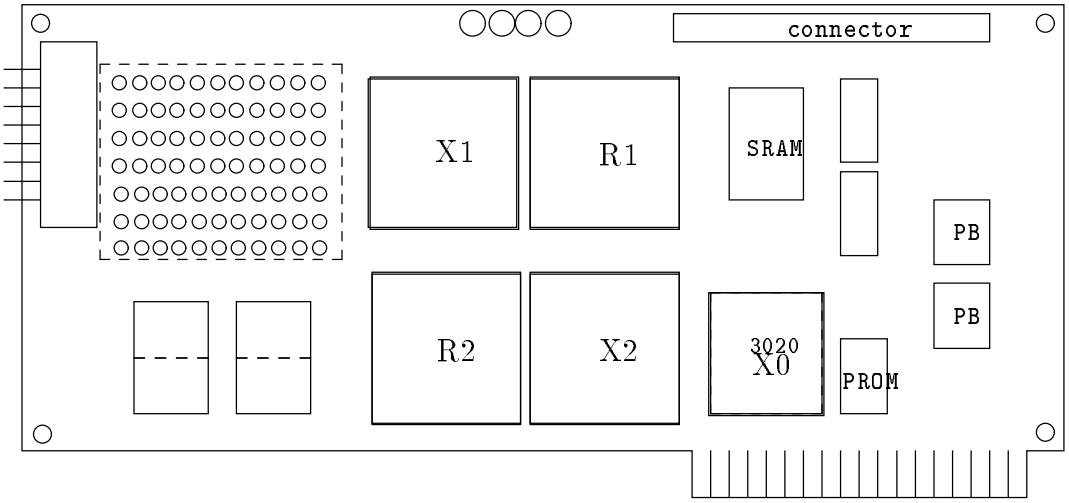

Figure 8: A prototype of the BORG Prototyping board

References

[1] A. El Gamal, "Protozone: The PC-Based ASIC Design Frame, User's Guide," Tech. Rep. SISL90-???, Stanford Information Systems Laboratory, Stanford University, Aug. 1990.

[2] F. Pirri and O. Bigalli, "A hardware programmable generic interface for the IBM PC," tech. rep., Dipartimento Di Ingegneria Elettronica of the Universita' Di Firenze, Italy, Sept. 1990.

[3] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[4] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems.* Benjamin Cummings, 1988.

[5] XILINX*: The Programmable Gate Array Data Book.* 2100 Logic Drive, San Jose, CA 95124, 1991.