Improved Hardware Accelerated FPGA Placement with Node Swap

Christian Fobel and Gary Gréwal
Computing and Information Science
University of Guelph
Guelph, Ontario, Canada
cfobel@uoguelph.ca, gwg@cis.uoguelph.ca

Andrew Morton
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
arrmorton@uwaterloo.ca

***Abstract*— Field Programmable Gate Arrays (FPGA) have become solutions of choice for embedded applications with small to medium production numbers. As a result, good CAD tools to support their use are in demand. This paper presents a solution to the FPGA placement problem. Some of the best solutions to date use iterative improvement heuristics such as simulated annealing. However,the run-times of these stochastic solvers becomes unacceptably long for performing placement on large FPGAs. Instead a deterministic iterative solver is proposed that is implemented in hardware. It implements a node-swap heuristic that starts from an initial random placement and iterates until it finds locally optimal solution. Initial results indicate speedups of 3–4 times over software.**

I. Introduction

A key advantage of Field-Programmable Gate Arrays (FPGAs) over full-custom and semi-custom devices is that they provide relatively quick implementation from concept to physical realization. However, the compilation times for designs, which are dominated primarily by *placement* and *routing*, are growing much more rapidly than the available computation power. For example, the time to compile current FPGAs can easily take hours or even days to complete for large 2-million gate chips. With 4-million gate chips on the horizon, long compile times may hurt the time-to-market advantage of FPGAs. Therefore, there is a need for the development of high-quality CAD tools that execute in a reasonable amount of CPU time, while still generating high-quality solutions.

In this paper, we focus on the *placement* phase of the FPGA-based design process. Placement is an NP-complete problem and one of the most time-consuming tasks in the automation of physical design. The most basic objective for FPGA placement is used: minimizing the total wire-length required to complete the routing.

We propose a fast, hardware-accelerated placement algorithm that employs a simple iterative heuristic that seeks to minimize the total wire length (interconnect distance) for the design. The algorithm begins by first creating an initial (random) placement. Then, on each iteration, it attempts to reposition each logic block on the FPGA in a way that reduces the total wire length required to connect (route) the blocks. The process repeats (iterates) until a local minima is found.

The next section briefly describes the placement problem and previous work in the area. Then in Section III, our first hardware-accelerated heuristic is described. Improvements to the heuristic are then explored followed by results in Section IV and conclusions.

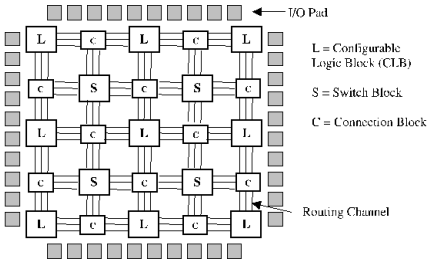

Fig. 1. Island Style FPGA

II. FPGA Placement

FPGAs are user-programmable integrated circuits that provide flexibility and reconfiguration advantages for supporting the design and production of digital systems. There is a variety of architectures available from different vendors, and although the exact structure of these FPGAs varies, all FPGAs consists of three fundamental components: (i) Logic blocks that are capable of implementing combinational/sequential logic functions; (ii) I/O blocks for communication with the outside world; and, (iii) Fixed, as well as programmable, routing resources used to realize all required interconnections between blocks.

In this paper, we assume that the FPGA is an *Island Style* architecture (see Fig. 1). The logic blocks in this architecture are referred to as *Configurable Logic Blocks (CLBs)* and are arranged as a symmetrical array. CLBs support the logic and storage elements of the circuit. General-purpose interconnection resources provide *wiring channels* which have a *Manhattan* geometry; i.e., the tracks are either horizontal or vertical. A *connection block* is used to connect a CLB to the routing channels via programmable connections. The pins of each CLB pass uninterrupted through the connection blocks and have the option of *fusing* to some channel segments. The *switch block* is a switch matrix that is used to connect wires in one channel segment to other wires.

Typically, implementing a digital circuit on an FPGA is a 3-step process. The first step involves partitioning of the circuit (described as a netlist) into logic blocks, in a way that allows each logic block to be implemented using one CLB. Next, each logic block is assigned to a specific CLB or I/O block. (The placement must be optimized so that the circuit can be successfully routed and signal delays meet any timing constraints.) Finally, an attempt is made to route the circuit

1-4244-1164-5/07/$25.00 ©2007 IEEE.

using the available routing resources.

As discussed, placement is a difficult, time-consuming step in the compilation process. Although fast routing algorithms that can find routes in seconds (or in some cases milliseconds) do exist (e.g., [4]), placement remains a significant problem. Consequently, a variety of approaches to placement have been proposed. In general, these approaches can be organized into three categories: *Partition-based* [5], *iterative improvement* heuristics [2] [3], and *analytic* methods [1].

In partitioning-based placement, a circuit is recursively bisected, minimizing the number of cuts of the nets that connect components between partitions, while leaving highly-connected blocks in one partition. Eventually, the partition size reaches a few blocks to obtain improvement by grouping the highly-connected blocks together. Partitioning-based placement algorithms are good from a "global" perspective, but they do not directly attempt to minimize wire length.

Analytical methods, on the other hand, typically employ a quadratic objective function, and iterate between two basic steps: solving sparse systems of linear equations and repartitioning. The primary advantage of analytical methods over partition-based methods is their speed: the system of linear equations can often be solved quickly. However, the inaccuracy of the quadratic model often leads to inferior solutions compared with other approaches.

In contrast, iterative improvement methods, start with initial placements and seek improvements by searching for small perturbations to the placements that result in better solutions. For example, simulated-annealing placement heuristics, like [2] [3], have achieved similar or higher quality solutions, compared with the other two types of tools. However, this improvement often comes at the expense of longer runtime.

III. Placement Heuristic

Before describing our solution, we define the placement problem. The placement problem is represented as a multi-graph: each edge connects two vertices and there can be multiple edges between vertices. Each vertex represent a CLB and each edge a connection between CLBs. The vertices are to be assigned to a grid, representing a position in the island architecture FPGA. The wire length l_{ij} is the orthogonal distance between vertices i and j. The objective of the placement heuristic is to minimize the total interconnect length L:

$$\min L = \left(\sum_{i=1 \cdots N} \sum_{j \neq i} l_{ij} \right).$$

Two versions of the hardware-accelerated placement tool were developed. The first of a simple node-move heuristic and the second improves by doing node-swaps. The node-move heuristic is described first, followed by the node-swap heuristic.

A. Node-Move Version

The node-move heuristic starts with a non-overlapping random assignment of CLBs to grid locations. The initial assignment for an example problem is depicted in Fig. 2. In each iteration, the heuristic attempts to find an improving move for each CLB. If after an iteration, the net change reduces the total interconnect length L, a new iteration is started. Otherwise, the heuristic terminates. The heuristic is summarized in Algorithm 1.

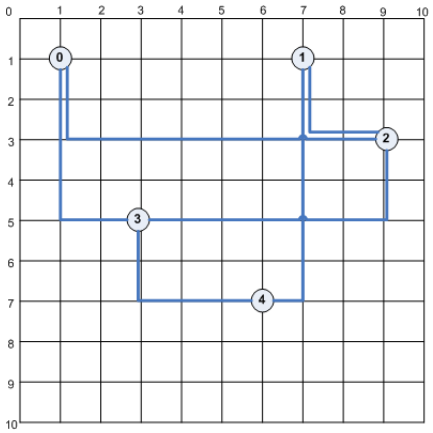

Fig. 2. Example Initial Placement

Algorithm 1: Placement Heuristic

```
generate random placement;
repeat
    ΔL = 0;
    for i = 1 to N do  ΔL += move(i);
until ΔL ≥ 0;
```

The improve(i) function searches a bounded box (neighbourhood) around CLB_i for a move that has a net decrease in L. It takes the first improving move found and is described by Algorithm 2. The neighbourhood size is a parameter of the heuristic: larger neighbourhoods result in potentially better solutions and smaller neighbourhoods are quicker to search.

Algorithm 2: Move CLB

```
calculate x and y ranges;
for x ∈ range[min,max] do
    for y ∈ range[min,max] do
        δ = 0;
        for j = 1 to N, j != i do
            if i,j connected then
                curr_dist = get_dist(i,j);
                new_dist = get_dist((x,y), j);
                δ += new_dist - curr_dist;
        if δ < 0 and !occupied(x,y) then
            update(i, (x,y));
            return δ;
```

The architecture of the hardware accelerator for the node-move heuristic is described next.

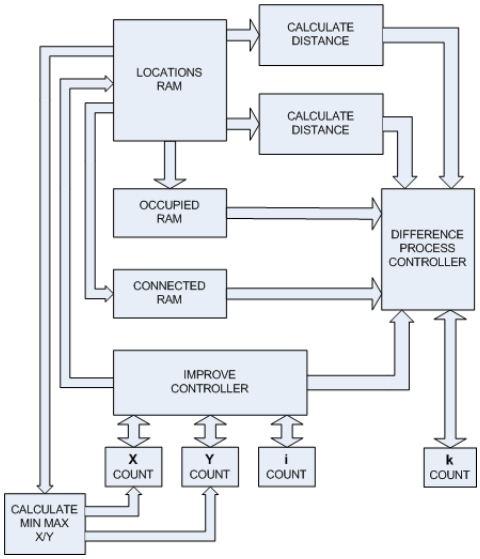

Fig. 3. Hardware Block Diagram

B. Hardware Accelerator

A block diagram of the hardware implementation of the placement heuristic is shown in Fig. 3. The coordinates of the CLBs are stored in the Locations RAM, and the wires connecting them are stored in the Connected RAM. The Location RAM is simply an array, indexed by CLB, of (x, y) coordinates. The Connected RAM has one entry per CLB pair (i, j) where $i > j$. An address encoder maps the CLB (i, j) to a location in the linear array. The entry for each CLB pair (i, j) indicates the width of the connection (number of wires). The Occupied RAM has one entry for each (x, y) grid location. If a location is occupied, it contains the CLB number, otherwise -1. The Improve controller is the main control for the accelerator. It selects candidate i for moving and iterates through the x, y permutations in i's neighbourhood. The Difference controller, checks the difference in total interconnect length L for each candidate move location. The Improve controller is responsible for halting the search once an iteration produces no improvement in L.

When the hardware accelerator for the node-move heuristic is synthesized for the Altera Stratix EP1S40, problems up to size 550 nodes (CLBs) can be handled. The Stratix EP1S40 has 41250 logic blocks and approximately 420 KB of memory. The maximum clock frequency of this design is 40MHz.

C. Node-Swap Version

The key change in the second implementation of the placement accelerator is that a CLB may now be moved to a location occupied by another CLB. Previously CLBs could only be moved to unoccupied locations within the search neighbourhood. This can still happen but CLB swaps are now also allowed. This expands the search space which can result in improved solutions, as is demonstrated by the results in Section IV. Algorithm 1 remains unchanged but Algorithm 2 is modified to allow swaps or moves. The modified pseudocode is listed in Algorithm 3.

TABLE I
TEST CASES

Test Case	Grid	CLBs	Connections	
			Sparse	Dense
1	9x9	20	27	190
2	25x25	150	1118	11175
3	34x34	280	3255	39060
4	41x41	410	6987	83845
5	47x47	550	13227	150975

Algorithm 3: Move or Swap CLB

```
calculate x and y ranges;
for x ∈ range[min,max] do
    for y ∈ range[min,max] do
        δ = 0;
        for j = 1 to N, j != i do
            if i,j connected then
                curr_dist = get_dist (i,j) ;
                new_dist = get_dist ((x,y), j) ;
                δ += new_dist - curr_dist;
        if occupied (x,y) then
            k = CLB at (x,y);
            for j = 1 to N, j != k do
                if k,j connected then
                    curr_dist = get_dist (k,j) ;
                    new_dist = get_dist ((x,y), j) ;
                    δ += new_dist - curr_dist;
        if δ < 0 then
            update(i, k) ;
            return δ;
```

It must now check whether the candidate position is occupied or not. If occupied, then it must determine the effect of moving that CLB into the location occupied by main CLB under consideration. This significantly increases execution time in software but can be done in parallel by the hardware.

IV. RESULTS

The two placement heuristics were tested using randomly generated test cases. The characteristics of these test cases are listed in Table I. For each test case there is a sparsely and densely connected version.

Both the node-move and node-swap heuristics were tested with each test case. Each heuristic was run 15 times for each test case, each starting from a different random initial placement. The average software and hardware times are reported in seconds in Table II. The software times were measured on a desktop with a 2.0GHz AMD Athlon X2 3800+ and 2GB of RAM running Windows under minimal system load.

TABLE II
HEURISTIC EXECUTION TIMES

(a) Sparse

Test	Node-Move			Node-Swap		
Case	HW	SW	Speedup	HW	SW	Speedup
1	1.66e-3	2.75e-3	1.66	1.88e-3	3.56e-3	1.89
2	0.28	0.44	1.61	0.30	0.72	2.42
3	1.36	2.21	1.63	1.60	3.714	2.31
4	3.64	5.90	1.64	5.02	11.65	2.32
5	7.26	11.14	1.56	10.52	26.46	2.51

(b) Dense

Test	Node-Move			Node-Swap		
Case	HW	SW	Speedup	HW	SW	Speedup
1	1.56e-3	5.33e-3	3.52	1.69e-3	5.03e-3	2.98
2	0.25	0.80	3.50	0.39	1.48	3.77
3	1.12	3.60	3.21	2.56	9.64	3.76
4	3.11	10.57	3.41	8.68	32.31	3.72
5	7.05	23.12	3.35	22.88	97.88	4.28

TABLE III
PLACEMENT QUALITY

(a) Sparse

Test	Length		Reduction		Solution Quality
Case	Move	Swap	Move	Swap	Swap vs Move
1	101	89	265	277	4.66%
2	21077	16009	29727	34795	17.05%
3	86560	65289	109248	130519	19.47%
4	229128	175810	274459	327777	19.43%
5	507518	419675	569116	656959	15.44%

(b) Dense

Test	Length		Reduction		Solution Quality
Case	Move	Swap	Move	Swap	Swap vs Move
1	1331	1242	1435	1524	6.21%
2	233782	221553	258797	271026	4.73%
3	1125887	1087213	1216525	1255199	3.18%
4	2913125	2830883	3104276	3186519	2.65%
5	6097374	5945336	6356506	6508543	2.39%

Both heuristics achieve greater speedups in the densely connected test cases than in the sparsely connected test cases. The improved speedup for the densely connected test cases is explained by the fact that the hardware implementation calculates the Manhattan distance between two nodes regardless of whether they are connected. The software implementation on the other hand only calculates the distance if they are connected.

The node-swap heuristic achieves speedups over its software implementation that are 10–30% greater than the node-move heuristic achieves. However the actual execution times are 1–4 times longer. So for execution time, the node-move heuristic is preferable to the node-swap heuristic. However the node-swap heuristic consistently finds solutions of better quality. Table III compares the solution qualities.

For sparsely connected problems, the difference in solution quality is significant: 5%–20%. For densely connected problems, the difference is not as notable: 2%–6%. Considering that all of the hardware execution times are less than a minute, it is probably advantageous to use the node-swap heuristic due to the improved quality of solution. For problems larger than 550 CLBs the node-move heuristic may be preferable based on its shorter execution time.

An approach to consider for further improvement of execution times is to pipeline the distance calculation and comparison. A distance calculation requires two differences and a summation. By pipelining, a speedup over current times of at least 3 is expected.

V. CONCLUSIONS

A compact hardware accelerator for the FPGA placement problem has been presented. It implements an iterative greedy move/swap heuristic that finds locally optimal solutions. The accelerators were synthesized for an FPGA to obtain size and time measurements. Initial results show moderate speedups of the hardware implementation over a software implementation. A design change is proposed to improve the accelerator: pipeline distance calculations. Instead of implementing the entire placement algorithm in hardware future work will examine a hardware/software adaptation of stochastic techniques.

REFERENCES

[1] C. J. Alpert, T. Chan, D. Huang, I. Markov, and K. Yan, "Quadratic Placement Revisited," ACM/IEEE Design Automation Conference, pp. 752-757, 1997.

[2] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," Proceedings of the International Workshop on Field Programmable Logic and Applications, pp. 213-222, 1997.

[3] V. Betz, J. Rose, and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, ISBN 0-7923-8460-1, 1999.

[4] A. DeHon, R. Huang, and J. Wawrzynek, "Hardware-Assisted Fast Routing," IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, CA, 2002.

[5] J. M. Kleinhans, G. Sigl, F. M., Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, " IEEE Transactions on Computer-Aided Design, 10(3):356-365, March, 1991.