

Placement for Modern FPGAs

Wai-Kei Mak

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan R.O.C.

Hao Li

Department of Computer Science and Engineering
University of North Texas
Denton TX, 76203, USA

Abstract—Modern FPGAs are not only increasing in size but have also become more complex with many new features not found in the previous FPGA generation. This demands more powerful physical design automation tools and better solutions for both old and new problems. In this paper, we describe a novel net-based force-directed FPGA placement algorithm. It outperforms the state-of-the-art FPGA performance-driven placement tool, VPR [9], with 10.7% shorter critical path delay and 11.5% shorter total wire length after routing over a set of MCNC benchmarks. In addition, we describe the new constrained I/O placement problem in modern FPGAs and present a powerful integer linear programming based approach to solve it. We applied our proposed method on a large set of benchmarks to show that the existing solution method implemented in Altera’s Quartus II tool suite is far from optimal.

I. INTRODUCTION

Placement directly affects the routability and the performance of a design on a FPGA. For a long time, VPR [9] has been the state-of-the-art timing-driven FPGA placement tool. VPR’s placement engine is based on simulated annealing. In [7], Maidee et al. proposed a timing-driven partitioning-based FPGA placement algorithm. The timing-driven partitioning-based algorithm is faster but still only achieves similar circuit delays as VPR on a set of benchmarks. Recently, there are a number of works that incorporate additional optimization such as logic replication and retiming to further improve the results of VPR. In [1], [4], logic replication is proposed as a post-processing step after placement by VPR to further improve circuit delay. Singh and Brown [10] proposed a post-placement retiming algorithm and showed how to modify VPR to make it “retiming aware”. Finally, Chen and Cong [3] presented a simultaneous placement and replication to minimize the longest path delay. These works required the changing of the netlist structure to improve the results of VPR. However, there has been no reported alternative placement algorithm which yields better circuit delays than VPR without changing the netlist structure.

In the first part of this paper, we describe a novel force-directed placement algorithm [6]. It is the first algorithm that outperforms VPR by a significant margin in terms of post-routing circuit delay and wire length. This result is even more remarkable considering that no timing analysis is carried out in our placement algorithm.

The placement of I/Os on a FPGA have become a non-trivial problem in today’s design because a FPGA can be connected to multiple devices employing different I/O standards. In the second part of this paper, we describe the constrained I/O placement problem and a powerful integer linear programming based approach to solve it [8].

II. NET-BASED FORCE-DIRECTED PLACEMENT

In this section, we propose a net-based force-directed placement algorithm for FPGAs. The algorithm performs placement in a top-down manner and it consists of three levels. The top level of our algorithm partitions the nets into clusters, and net-cluster level floorplanning is performed using simulated annealing to optimize the total netlength. The intermediate level of our algorithm computes a coarse net placement using a force-directed method. The bottom level of our approach achieves the detailed placement also using force-directed method where the forces on nets determine the positions of the logic blocks in this phase.

A. Net Cluster Floorplanning

In the top level of our algorithm, a net dependency graph is constructed from the netlist and clique partitioning is carried out on this graph. A *net clique*, also called a *net cluster*, is a complete subgraph of the net dependency graph. A net cluster indicates a set of strongly connected nets. Intuitively, it is preferable to place and route nets belonging to the same clique in nearby locations because we can optimize the total netlength and the performance of the circuit. Once the net dependency graph is partitioned into net clusters, we place the net clusters in such a way that the overall interconnect is minimized.

The net cluster level floorplan is obtained by simulated annealing using the sequence-pair representation [5]. Since the number of net clusters is smaller than the number of logic blocks, the time needed to find a net cluster level floorplan is much less than the time required to find a floorplan at the logic block level which is very desirable.

The cost function of a floorplan, \mathcal{F} , is a weighted sum of the interconnections between clusters, and the interconnections between clusters and I/O pins on the boundaries of the target

device:

$$Cost(\mathcal{F}) = \sum_{i \neq j} k_1 \cdot D(C_i, C_j) + \sum_i k_2 \cdot N_i \cdot I(C_i) \quad (1)$$

where $D(C_i, C_j)$ is the Manhattan distance between the geometrical centers of the bounding boxes of clusters C_i and C_j , $I(C_i)$ is the shortest distance between the geometrical center of C_i and the boundaries of the device, N_i is the number of logic blocks in C_i which have connections with I/O pins. The factors k_1 and k_2 in Equation 1 account for the overhead for routing. They are obtained empirically and can be adjusted according to the routing resources availability of the FPGA.

B. Net-level Placement

The intermediate level of our algorithm is to perform a coarse net level placement using a force-directed scheme. We use two types of forces. The main force is the *attractive force* which obeys Hooke's law. It pulls nets connected in the net dependency graph to closer positions. We also apply a *repulsive force* to avoid net cluster overlaps. We have to find the equilibrium positions for the objects. In general, the attractive force on object i due to object j is computed as follows:

$$\vec{F}_{(i,j)}^a = -K_{(i,j)}^a \cdot (\vec{p}_i - \vec{p}_j) \quad (2)$$

where $K_{(i,j)}^a$ is an analogous attractive coefficient, \vec{p}_i and \vec{p}_j are the position vectors of objects i and j , respectively.

To prevent two objects from overlapping, we introduce some form of repulsive force. The repulsive force has an electrostatic-like characteristic, which is strong at close distances and weak at long distances. So, whenever two objects are getting too close, the repulsive force will be very strong to push them apart. The repulsive force on object i due to object j is computed as:

$$\vec{F}_{(i,j)}^r = K^r \cdot (r/d)^2 \frac{\vec{p}_i - \vec{p}_j}{|\vec{p}_i - \vec{p}_j|} \quad (3)$$

where K^r is the repulsion constant, r denotes the sum of $R_i + R_j$ where R_i and R_j are the radii of objects i and j , respectively, and d is the distance between the two objects. In order to find the equilibrium positions for a set of objects, we need to find a position for each object such that the total force on object i , $\vec{F}_{total}(i)$, is zero. $\vec{F}_{total}(i)$ is computed as shown in Equation 4:

$$\vec{F}_{total}(i) = \sum_{j=1, i \neq j}^n [\vec{F}_{(i,j)}^a + \vec{F}_{(i,j)}^r] \quad (4)$$

The equilibrium position for object i , $\vec{p}_{(i,e)}$, can be found as follows:

$$\vec{p}_{(i,e)} = \vec{p}_i - k_e \vec{F}_{total}(i) \Big/ \frac{\partial \vec{F}_{total}(i)}{\partial \vec{p}} \quad (5)$$

where k_e is a equilibrium constant, and \vec{p}_i is the original position of object i . We perform this computation on all

objects one by one until the equilibrium positions of all objects are found.

After computing a net cluster floorplan, we use the force-directed method to obtain a coarse net placement. The force on a net n_i due to a net cluster C_l is given by Equation 6.

$$\vec{F}(n_i) = \sum_{(n_i, n_j) \in E} \vec{F}^a(n_i, n_j) + \sum_{\substack{i \neq j \\ n_j \in C_l}} \vec{F}^r(n_i, n_j) + k_l \vec{F}^a(n_i, C_l) \quad (6)$$

The first term is the attractive force between net n_i and all nets n_j which are adjacent to n_i in the net dependency graph G . The second term describes the repulsive force between nets in the same net cluster. The last term is the force to maintain net n_i in the region allocated for the corresponding net cluster C_l . The factor k_l is the number of nets connected with n_i but do not belong to C_l . The equilibrium positions of the nets can be calculated using Equation 5.

C. Logic Block Placement

In this phase, an iterative force-directed scheme is then carried out on the logic blocks until the equilibrium positions are found. The force on a logic block s_i is computed as:

$$\begin{aligned} \vec{F}(s_i) &= \sum_{n_j \in N_i} \vec{F}^a(s_i, n_j) + \gamma \sum_{I/O_j \in P_i} \vec{F}^a(s_i, I/O_j) \\ &+ \delta \sum_{s_j \in S_i} \vec{F}^r(s_i, s_j) \end{aligned} \quad (7)$$

The first term is an attraction force between logic block s_i and the nets to which s_i is connected with. N_i denotes the set of nets connected with this logic block. The second term is also an attraction force. If logic block s_i is connected with I/O blocks, this force attracts s_i to the chip boundary. Here, we choose the closest side of the chip to s_i as "chip boundary". P_i denotes the set of I/O pins that are connected with s_i , and γ is a weighting factor bigger than 1 to indicate the priority of interconnections with I/O pins. The last term is a repulsive force between s_i and all logic blocks that are adjacent to it. S_i denotes the set of logic blocks connected with s_i .

Once the positions of all the logic blocks are stable, we will place them on the FPGA chip. Since the locations of CLBs are represented by integers, we convert the coordinates of the logic blocks into integers. If there are more than one logic block assigned to the same CLB location, we will place the outstanding ones into nearby available CLBs.

III. CONSTRAINED I/O PLACEMENT

In this section, we consider the constrained I/O placement problem when a FPGA is connected to multiple devices employing different I/O standards. Today's FPGAs that support multiple I/O standards all use a banked I/O organization similar to the one shown in Fig. 1. The I/O blocks of a bank are served by a single V_{ref} voltage and a single V_{cco} voltage. However, different I/O standards have different voltage

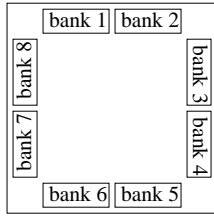


Fig. 1. Banked I/O organization.

requirements. For example, Table 1 lists the requirements of all the I/O standards supported by Altera's Stratix devices. Note that the voltage requirements of the input and output objects can be different even for the same standard. And a bidirectional object has the combined requirements of an input object and an output object of that standard.

TABLE I
I/O STANDARDS SUPPORTED BY STRATIX AND THEIR VOLTAGE REQUIREMENTS.

I/O Standard	Output V_{CCO} (V)	Input V_{CCO} (V)	Input V_{REF} (V)
LVTTL	3.3	3.3	N/A
LVC MOS	3.3	3.3	N/A
2.5V	2.5	2.5	N/A
1.8V	1.8	1.8	N/A
1.5V	1.5	1.5	N/A
3.3-V PCI	3.3	3.3	N/A
3.3-V PCI-X 1.0	3.3	3.3	N/A
LVDS	3.3	3.3	N/A
LVPECL	3.3	3.3	N/A
3.3-V PCML	3.3	3.3	N/A
HYPERTRANSPORT	2.5	2.5	N/A
Differential HSTL	1.5	N/A	0.75
Differential SSTL	2.5	N/A	1.25
GTL	3.3	N/A	0.8
GTL+	3.3	N/A	1.0
1.5-V HSTL I & II	1.5	N/A	0.75
1.8-V HSTL I & II	1.8	N/A	0.9
SSTL-18 I & II	1.8	N/A	0.9
SSTL-2 I & II	2.5	N/A	1.25
SSTL-3 I & II	3.3	N/A	1.5
AGP(1X & 2X)	3.3	N/A	1.32
CTT	3.3	N/A	1.5

We observe that each I/O bank can be configured with at most one V_{CCO} voltage and one V_{REF} voltage. Depending on the V_{CCO} and V_{REF} voltage configuration of a bank, some classes of I/O objects can be put in the bank while other classes cannot. So we propose a 0-1 integer linear programming based approach to solve the constrained I/O placement problem.

First, we classify the I/O objects into four major types. Type *A* I/O objects are those that require a V_{CCO} voltage but not a V_{REF} voltage. Type *B* I/O objects are those that require a V_{REF} voltage but not a V_{CCO} voltage. Type *C* I/O objects are those that require both a V_{CCO} voltage and a V_{REF} voltage. Type *D* I/O objects are those that require neither a V_{CCO} voltage nor a V_{REF} voltage.

We define a binary decision variable p_i^k for each bank k such that p_i^k is 1 if bank k is configured with a V_{CCO} voltage equals to V_{C_i} , and p_i^k is 0 otherwise. Similarly, we define a binary decision variable q_j^k for each bank k such that q_j^k is 1 if and only if bank k is configured with a V_{REF} voltage equals to V_{R_j} .

And we define

$$\begin{aligned}
 a_i^k &= \text{no. of class } A_i \text{ I/O objects assigned to bank } k \\
 b_j^k &= \text{no. of class } B_j \text{ I/O objects assigned to bank } k \\
 c_{ij}^k &= \text{no. of class } C_{ij} \text{ I/O objects assigned to bank } k \\
 d^k &= \text{no. of class } D \text{ I/O objects assigned to bank } k
 \end{aligned}$$

Inputs of the constrained I/O placement problem are:

$$\begin{aligned}
 U^k &= \text{no. of regular user I/O pins in bank } k \\
 R^k &= \text{no. of } V_{REF} \text{ pins in bank } k \\
 |A_i| &= \text{no. of class } A_i \text{ I/O objects to be placed} \\
 |B_j| &= \text{no. of class } B_j \text{ I/O objects to be placed} \\
 |C_{ij}| &= \text{no. of class } C_{ij} \text{ I/O objects to be placed} \\
 |D| &= \text{no. of class } D \text{ I/O objects to be placed}
 \end{aligned}$$

The complete ILP is shown below.

min z

$$\text{s.t. } \sum_{i=1}^4 a_i^k + \sum_{j=1}^7 b_j^k + \sum_{(i,j) \in F} c_{ij}^k + d^k \leq U^k \quad \forall k = 1, \dots, 8 \quad (8)$$

$$\sum_{k=1}^8 a_i^k = |A_i| \quad \forall i = 1, \dots, 4 \quad (9)$$

$$\sum_{k=1}^8 b_j^k = |B_j| \quad \forall j = 1, \dots, 7 \quad (10)$$

$$\sum_{k=1}^8 c_{ij}^k = |C_{ij}| \quad \forall (i, j) \in F \quad (11)$$

$$\sum_{k=1}^8 d^k = |D| \quad (12)$$

$$a_i^k \leq \min(|A_i|, U^k) \cdot p_i^k \quad \forall i = 1, \dots, 4; k = 1, \dots, 8 \quad (13)$$

$$b_j^k \leq \min(|B_j|, U^k) \cdot q_j^k \quad \forall j = 1, \dots, 7; k = 1, \dots, 8 \quad (14)$$

$$c_{ij}^k \leq \min(|C_{ij}|, U^k) \cdot p_i^k \quad \forall (i, j) \in F; k = 1, \dots, 8 \quad (15)$$

$$c_{ij}^k \leq \min(|C_{ij}|, U^k) \cdot q_j^k \quad \forall (i, j) \in F; k = 1, \dots, 8 \quad (16)$$

$$\sum_{i=1}^4 p_i^k = 1 \quad \forall k = 1, \dots, 8 \quad (17)$$

$$\sum_{j=1}^7 q_j^k = 1 \quad \forall k = 1, \dots, 8 \quad (18)$$

$$a_i^k, b_j^k, c_{ij}^k, d^k \geq 0 \quad \forall i, j, k \quad (19)$$

$$p_i^k, q_j^k \in \{0, 1\} \quad \forall i, j, k \quad (20)$$

(8) ensures that the number of I/O pins consumed in a bank does not exceed the number of pins available. (9)–(12) ensure that all I/O signals of each class are assigned to I/O banks and no I/O signals are left unassigned. (13)–(16) correspond to the allowance constraints. For example, class A_i I/O signals are only allowed in bank k if the binary decision variable p_i^k is 1 (i.e., bank k is configured with a V_{CCO} voltage of V_{C_i}). (13)

does not limit the number of class A_i I/O signals assigned to bank k when p_i^k is 1 since the number of class A_i I/O signals assigned to a bank can neither exceed the total number of class A_i I/O signals nor the number of available I/O pins in the bank. On the other hand, it ensures that the number of class A_i I/O signals assigned to bank k must be zero when p_i^k is 0. Similar restrictions for class B_j I/O signals and class C_{ij} I/O signals are enforced by constraints (14) to (16). (17) and (18) ensure that each bank is assigned exactly one V_{cc0} voltage and one V_{ref} voltage, respectively. If there are different preference or cost associated with assigning different classes of I/O signals to a bank, the objective function z can be set equal to the sum of the assignment cost of the solution.

IV. EXPERIMENTAL RESULTS

Firstly, we implemented the proposed force-directed placement algorithm and compared it with VPR [9], a well-known place and route suite developed by the University of Toronto. Each circuit is firstly optimized in SIS using “script.rugged” and then technology-mapped to 4-input look-up tables (LUTs) with Flowmap. Next, VPACK is used to pack the circuit into CLBs. We first run VPR to place and route each design in timing-driven mode. We set the size of the target FPGA to be 80x80 CLBs and we use the file “4lut_sanitized.arch” as the architecture file. We also obtain the channel width used by VPR for each circuit i , denote by W_i . Next, we run our algorithm to generate the placement. While performing re-entrant routing on circuit i using VPR, we specify the option “-route_chan_width W_i ”, where W_i is derived previously. The experimental results are shown in Table II where “D1” denotes the total net delay, “D2” denotes the critical path delay, “W” denotes the channel width, and “T” denotes the CPU time on placing the circuit.

TABLE II
COMPARISON WITH VPR

Ckt	VPR				Ours				Improvement	
	D1 (ns)	D2 (ns)	W	T(s)	D1 (ns)	D2 (ns)	W	T(s)	D1 (%)	D2 (%)
c2670	94.4	99.0	6	27.9	83.6	93.1	6	29.2	11.5	6.0
c3540	84.3	94.9	9	25.5	77.2	87.1	9	23.5	8.4	8.2
c5315	93.3	100	11	47.2	84.8	92.1	11	41.9	9.1	7.9
c6288	154	179	5	31.9	133.7	169.2	5	27.2	6.7	5.5
c7552	147	163	7	42.8	126	138	7	48.9	14.3	15.3
dalu	92.8	96.8	9	21.4	70.7	74.7	9	23.8	23.8	22.3
des	124	127	7	225.3	108.5	113.8	7	201.3	12.5	10.4
i10	167	185	10	118.6	142	161	10	111.5	14.9	13.0
i8	84.2	89.4	12	43.8	77.1	79.5	12	34.1	8.4	11.1
k2	66.8	74.1	11	24.2	61.3	67.5	11	16.5	8.2	8.9
pair	83.1	93.2	7	43.7	75.6	84.6	7	39.1	9.0	9.2
Avg.									11.5	10.7

On average, compared with VPR, our algorithm reduces the total net delay and the critical path delay by 11.5% and 10.7%, respectively. As to the runtime, our algorithm is slightly faster than simulated annealing based VPR placer. In addition, the placement generated by our program does not require extra routing tracks for VPR to perform re-entrant routing.

Secondly, we compared our 0-1 ILP-based I/O placement approach with an industrial FPGA CAD tool, Altera Quartus

TABLE III

RESULTS OF EXPERIMENT. (* SEE EXPLANATION IN THE TEXT.)

Batch	#I/O objects	Quartus II 4.1			Our ILP Approach		
		#successfully placed cases	run-time(sec)		#successfully placed cases	run-time(sec)	
			avg.	max.		avg.	max.
I	723 - 808	85	104	179	100	1.41	25.75
II	803 - 928	69	103	158	99	3.96	120*
III	883 - 968	24	54	165	90	17.8	120*

II 4.1, on a large number of test cases. Our target device is an Altera Stratix EP1S80F1508 FPGA which has twelve I/O banks. We used three batches of test cases each containing a hundred problem instances generated by a generator provided by Altera. We used lp_solve5.1[2] to solve the corresponding ILPs. Since feasible instances usually can be solved very quickly, we set a time limit of 120 seconds for the integer linear programming solver and treated those cases which cannot be solved within the time limit as infeasible. (Out of 300 instances, timeout only occurred four times.) The results are shown in Table III. We found that the solution method implemented by Quartus II is far from optimal. Quartus often reported feasible cases as infeasible, and the situation got worst when the number of I/O objects was increased.

V. CONCLUSIONS

We presented a net-based force-directed placement algorithm and a ILP-based I/O placement algorithm for modern FPGAs. Our placement algorithm outperformed the well-known VPR algorithm in both wirelength and critical path delay. And our I/O placement algorithm was able to find a legal I/O placement for many instances for which an industrial tool has failed.

REFERENCES

- [1] G. Beraudo and J. Lillis, “Timing Optimization of FPGA Placements by Logic Replication”, in *Proc. of Design Automation Conf.*, pp. 196-201, 2003.
- [2] M. Berkelaar, K. Eikland, and P. Notebaert, *lp_solve*, available at http://groups.yahoo.com.tw/group/lp_solve.
- [3] G. Chen and J. Cong, “Simultaneous Timing-Driven Placement and Duplication”, in *Proc. of International Symp. on Field-Programmable Gate Arrays*, pp. 51-59, 2005.
- [4] M. Hrkic, J. Lillis, and G. Beraudo, “An Approach to Placement-Coupled Logic Replication”, in *Proc. of Design Automation Conf.*, pp. 711-716, 2004.
- [5] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair,” *IEEE Transactions on Computer Aided Design*, vol.15, pp. 1518-1524, Dec. 1996.
- [6] H. Li, S. Katkooi, and W.K. Mak, “Force-directed performance driven placement algorithm for FPGAs,” in *Proceedings ISVLSI 2004*, pp. 193-198, 2004.
- [7] P. Maidee, C. Ababei, and K. Bazargan, “Fast Timing-driven Partitioning-based Placement for Island Style FPGAs”, in *Proc. of Design Automation Conf.*, pp.598 -603, 2003.
- [8] W.K. Mak, “Modern FPGA Constrained Placement”, in *Proc. of Asia South Pacific Design Automation Conf.*, pp. 779-784, 2005.
- [9] A. Marquardt, V. Betz, and J. Rose, “Timing-Driven Placement for FPGAs”, in *Proc. of International Symp. on Field-Programmable Gate Array*, pp. 203-213, 2000.
- [10] D.P. Singh and S.D. Brown, “Integrated Retiming and Placement for Field Programmable Gate Arrays”, in *Proc. of International Symp. on Field-Programmable Gate Array*, pp. 67-76, 2002.