RegPlace: A High Quality Open-source Placement Framework for Structured ASICs

Ashutosh Chakraborty, Anurag Kumar, David Z. Pan
ECE Dept. Univ. of Texas at Austin, Austin, TX 78712
{ashutosh, anurag}@cerc.utexas.edu, dpan@ece.utexas.edu

ABSTRACT

Structured ASICs have recently emerged as an exciting alternative to ASIC or FPGA design style as they provide a new trade-off between the high performance of ASIC design and low non-recurring engineering (NRE) costs of FPGA design. To fully utilize the benefits of structured ASICs, key physical design stage like placement should be made aware of modularity of their architecture. In this work, we propose a novel solution to placement for structured ASICs. In particular, we propose creation of intermediate virtual platform to exploit the regularity of structured ASIC, Integer Linear Program and network flow formulations for satisfying constraints associated with typical structured ASIC clock architectures. A preliminary version of this work recently won the structured ASIC placement contest by eASIC [1]. Our placer achieves 35% and 5% wirelength improvement over other placers and can place a design of 1 million cells in approximately 4 hours.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

General Terms

Algorithms, Design

Keywords

Placement, Structured ASIC, FPGA, Regular ASIC, Global Placement, legalization

1. INTRODUCTION

Skyrocketing costs and increasing variability associated with an ASIC design flow and unacceptable power and delay penalty associated with FPGA design flow have forced semiconductor companies to look for alternatives. One viable alternative that has emerged over time is the use of *structured* ASICs. Structured ASICs provide an exciting middle-ground between high performance of ASIC designs and short time-to-market FPGA designs. They exploit the fact that not all mask-layers provide equal value for the customers and these layers can be pre-fabricated amortizing their cost over multiple designs [2]. Structured ASIC flow is much simpler than that for traditional ASICs because majority of deep submicron issues such as signal integrity, power grid optimization, low skew clock tree distribution are already taken care of by the structured ASIC vendors. Structured ASICs can be used to implement designs consisting of millions of gates in contrast to FPGAs which can implement designs with much lesser number of gates. There are a wide variety of structured ASIC architectures, but all of them have a fundamental repeated logic element called *tile* which may contain pre-defined combinational logic, small RAM, and registers [3].

To fully utilize the benefits of structure ASICs, tools for high quality placement and routing need to be developed. Placement for structured ASIC requires cells to be mapped exactly on a compatible site, similar to the case for FPGA. However, since the problem size of structured ASIC can be an order of magnitude bigger than FPGA [2], the existing FPGA tools cannot be used for structured ASICs. Not only does the placer have to handle millions of cells along with the site compatibility requirement, the task is made much more difficult due to the clocking schemes in structured ASICs. The clocks are built with pre-allocated resources to provide low skew clocking and simplified design flow. This restricts the number of clocked elements which can be placed in proximity of each other. In this work we present RegPlace [4], a high quality open source placement tool for structured ASICs which can deal with the above mentioned legality and clock constraints.

The rest of the paper is organized as follows: We describe the intricacies of placement for structured ASIC in Section 2. Section 3 discusses the previous work done and our key innovations. We present our placement solution in Section 4. The efficacy of our solution is demonstrated by experimental results in Section 5. Section 6 concludes our paper.

2. PROBLEM DESCRIPTION

As discussed earlier, large problem size, site compatible requirements, whitespace requirement for routing tracks and strict clock constraints make the task of placement in structured ASICs very challenging. In this work, we propose a solution to the placement problem of structure ASICs. We base our work on the architecture of the popular Nextreme line of structured ASICs from eASIC Corporation [1]. Nextreme is the most *structured* of structured ASIC solutions with only one via layer available for customization. However, our formulations are very generic and can be applied to any structured ASIC.

There are four types of cells in Nextreme line of struc-

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.

tured ASICs. They are SRAM programmable Logic cells (LCELLS), flip flops (DFF), registers (REG) and memories (BRAM). In the rest of the paper, we will refer to these types of cells with their short names and all these types will collectively referred to as *cells*. The placement problem requires that a cell can only be placed on a location which is reserved for that type of cell. In the basic repeating *tile* of Nextreme architecture, the space reserved for LCELL, DFF, REG and BRAM are as shown in Figure 1 (figure is only for illustration and not drawn to scale). There are 36 LCELL and 24 DFF columns in each tile. Each of these columns accommodate 64 LCELLs and DFFs respectively. A total of 4 REG and 1 BRAM can be accommodated in each tile. This tile repeats all over the chip with some horizontal and vertical inter-tile whitespace between adjacent tiles. Depending on the size of the netlist to be placed, the number of times these tiles need to be repeated can be configured. Table 1 shows the details of two such configured platforms along with the maximum cells of each type that can be accommodated in them.

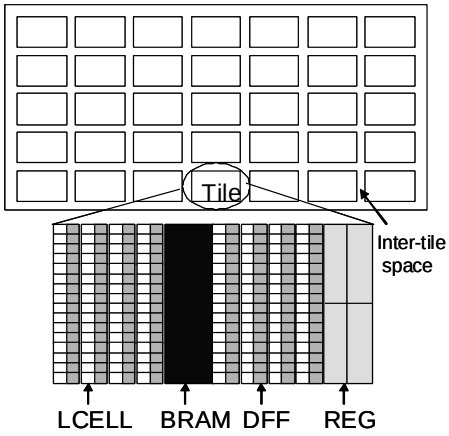

Figure 1: Structured ASIC platform.

Table 1: Platform Characteristics

Plat.	Max LCELL	Max DFF	Max REG	Max BRAM	Intertile Space	
					Horz	Vert
A	1M	675k	1760	440	8.00	6.45
B	1.2M	811k	2112	528	8.00	8.92

The sizes of each LCELL, DFF, REG and BRAM in the mapped design netlist are 1x1, 1x1, 8x32 and 16x64. These cells are to be placed according to the reserved column locations described above. The presence of these four different types of cells require that the global placement ensure that the density requirement for each type of cells is met. The presence of whitespaces and interleaving of space for each of the cell types makes the physical location available for each type of cell very discrete.

Because of architectural constraints, only a certain number of different clocks can go inside a tile. For example, in Figure 1, each tile can have N(=4 in our benchmarks) different clocks. Moreover, all the cells in one column of DFF can have only one clock type. The clock constraint on placement has big impact on final placement because clock violations may require that the violating cells be reallocated to another region far away from their optimal location.

3. PREVIOUS WORKS AND OUR CONTRIBUTION

There are several commercial and academic placers for ASIC designs e.g. [5] [6] [7] [8] [9]. These placers implement sophisticated mathematical or min-cut formulations to generate very good quality results for ASIC designs. However, due to site compatibility and hard clock constraints they cannot be directly used for placement of structured ASIC designs. Nevertheless, we note that if the key algorithms in these placers can be somehow used to solve the structured ASIC placement problem, the solutions would be of good quality. Another class of placement tools which are designed for FPGAs can take care of site constraints. However, due to smaller sizes of FPGA based designs, the existing FPGA placement approaches such as [10] [11] rely on slow algorithms like simulated annealing which cannot scale to problem sizes frequently encountered in structured ASIC designs.

There is limited research in the domain of placement for structured ASICs. The work in [12] only addresses incremental placement issues in structured ASICs. Industries are currently using existing row-based placers with product specific legalizers or heuristics [13]. However, there is a dearth of tools which can handle the clock constraints or exploit the modularity of structured ASIC platforms.

Our main technical contributions in this work are

- We propose row-based placer friendly virtual platform generation. This method achieves even density distribution and much faster placement.
- An integer linear program (ILP) formulation is proposed to satisfy clock constraints on the number and type of clocks that can appear in a tile as well as each column of a tile.
- A detail placement flow is proposed specifically for tile based structured ASIC architectures.

4. OUR PLACEMENT FLOW

Our complete placement flow is depicted in Figure 2. We outline the major steps here while the highlights of each step are discussed in next sections. In the first phase, given a design netlist and physical platform specification, we transform them into virtual netlist and virtual platform. This key step mitigates the problems arising due to severely discretized space available to the placer. A high quality row-based placer is run to place the virtual netlist on the virtual platform which generates an initial solution for our problem. In the second phase, we transform the placement solution back to the real platform while minimizing the impact on solution quality due to this transformation. The third stage performs the key step of satisfying clock and density requirement at the platform level using efficient mathematical formulation. This is followed by intra-tile clock assignment and perform aggressive wirelength reduction while maintaining the site legality of the solution. The key highlights of the above stages are presented below.

4.1 Virtual Platform Generation

The physical space available for placement of each type of cell is very discrete in our placement problem. One obvious method to take care of this is through the use of blockages. Though some existing placement tools do have the capability to consider blockages, we cannot specify our placement

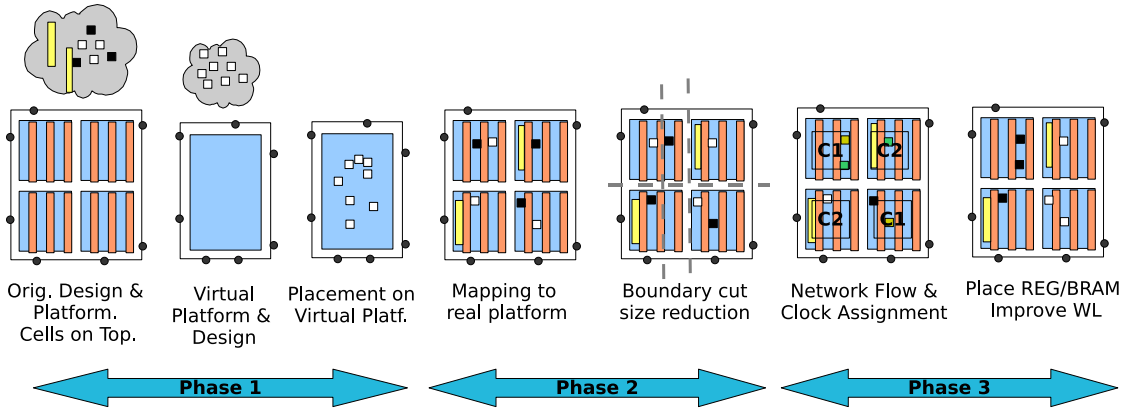

Figure 2: Flow of RegPlace. Different colored columns on platform represent different compatible sites.

problem with blockages because blockages for one type of cell can be valid site for another type of cell. Further, the solution quality and the time required by placers degrade significantly when considering several thousands of blockages. To overcome this challenge, we generated a virtual platform and corresponding virtual netlist. The virtual platform is a physically shrunk copy of the real platform with only the reserved space for one type of cell adjacent to each other. Since majority of the cells are of the type LCELLs, we made the virtual platform by stacking together all spaces reserved for LCELLs. In other words, the inter-tile spacing and the space reserved for DFFs, REGs, and BRAMs is removed leaving a contiguous virtual platform which has nearly the same height as the real platform but nearly 25% the width of real platform. The virtual nelist is generated by forcing the cell size of each type of cell to be equal to that of an LCELL. A standard row-based placer can now be exploited to place the virtual netlist onto the virtual platform. In the general case, the space available in the virtual platform may not be sufficient to contain the virtual netlist. In such a case, we expand the virtual platform horizontally until its size is at least 10% larger than the virtual netlist's space requirement to allow sufficient whitespace for the placer.

4.2 Transforming Virtual Placement Solution

The results of global placer on virtual platform needs to be mapped back to the real platform. We perform this step by inserting whitespace corresponding to the blockages for LCELLs. This step, which is depicted in Figure 3, can cause large increase in wirelength (WL) because several nets which were earlier small would be elongated by the amount of whitespace inserted. Note that in Figure 3, we assume only 4 columns of LCELL instead of 24 to avoid cluttering the diagram. To reduce this impact, we perform cut minimization on cells immediately lying on the boundary of the inserted whitespace.

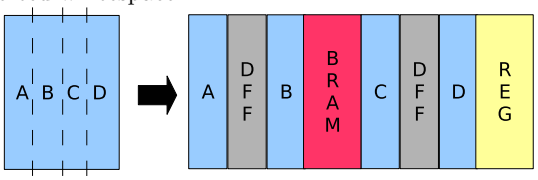

Figure 3: Example of shifting of LCELL columns to accommodate other cell types in between. Columns A, B, C and D spread to reform the tile structure

Since the initial placement is performed on a continuous virtual platform, all the cells get placed in a unique tile as a result of mapping the virtual placement solution on to the real platform. In other words, no cell occupies the spaces reserved for DFF, REGs and BRAMs as well as the spacing between individual tiles.

4.3 Chip Level Density & Clock Legalization

In the placement obtained from previous step, under the assumption that all cells are of the same type, each tile has density less than one i.e. they can be placed without overlap. However, for all the cells to be placed in their matching cell type, each tile's density for *each* type of cell should be less than one. It is easy to see that if, after all the cells have been assumed to be of type LCELL, there is no density violation, then, there can never be any density violation if some of the cells are turned back into their own type (thus leaving empty space in the space for LCELLs). Due to this reason, we never need to consider density of LCELLs in our approach and only the DFFs, BRAM and REG file need to be considered.

Satisfying DFF density: The first step for DFF density and clock constraint is to determine which set of clocks will be present in each tile under the constraint that no more than N different clocks exist per tile (N=4 in our implementation). Our placer formulates this problem as an ILP as follows:

In a design with T tiles and total of C clocks, let boolean variable E_{ti} denote the existence of clock type i in tile t. The effort of removing k_i cells of clock type i from a tile t is $(1 - E_{ti}) \times k_i$. The contribution of tile t to the cost function is the total effort to clean up the cells of a clock type which cannot be placed in the tile which is $\sum_{i=1}^{C}(1 - E_{ti}) \times k_i$. The complete cost function is simply addition of the above cost function for each tile. To guarantee enough space for all the cells of each clock type, few more constraints need to be put.

$$\text{Minimize} \quad \sum_{t=1}^{T}\left(\sum_{i=1}^{C}(1 - E_{ti}) \times k_i\right)$$

Subject to :

$$\sum_{i=1}^{C} E_{ti} \leq N \quad \forall t \in (1, T)$$

$$\sum_{t=1}^{C} E_{ti} \geq Min_i \quad \forall i \in (1, C)$$

$$E_{ti} \in 0, 1$$

where Min_i is the minimum number of tiles required for the DFF with clock i for the legalization to be possible. The above ILP formulation has O(T×C) boolean variables, O(T) constraints for guaranteeing no more than N clocks in each tiles, and O(TC) constraints to guarantee enough space for all cells. Considering that T is in hundreds and N is in tens, the above ILP formulation can be solved within seconds.

Once the clock assignment is fixed, we formulate the problem of determining how many DFFs to move among various tiles as a min-cost network flow problem with capacity constraints. The network flow is constructed by connecting a tile congested w.r.t. DFFs with the four tiles around it. The maximum capacity of any edge is taken as the minimum of the number of DFFs that need to exit a congested tile and the number of DFFs which can move into neighboring tile without congesting it. In our implementation, the cost of moving a cell out of its current tile is 1. Thus the network flow solution minimizes the *number of DFF* that need to move for satisfying density constraints. It is possible to extend our work by modeling the cost as change in WL due to moving a DFF which corresponds to a solving density constraints with least movement. At this stage, each tile satisfies density constraints w.r.t. DFFs.

Satisfying REG Density: Since each tile can have at most 4 REGs, and the number of REG cells to be placed are usually very small (several hundreds), it is possible to solve their placement globally. For T tiles, there are 4T valid places for REGs. We cast this selection as an assignment problem which can be solved efficiently using Munkres algorithm [14] whose efficient implementation is $O(n^3)$ complexity. The cost of placing a REG into a valid position is modeled as the wirelength(WL) of all the nets incident to the REG cell in that position. The solution to the assignment problem not only identifies which REG goes to which tile, but also fixes its position to a valid location in the tile. Even for the biggest benchmark and platform, this step takes less than 30 seconds CPU time.

Satisfying BRAM Density: The case for BRAM is exactly the same as that of REGs and can be solved by assignment problem readily. Notice that owing to very small number of BRAMs and only one space per tile, the size of assignment problem for BRAM is an order of magnitude smaller than that for REGs and is solved within 10 seconds for biggest benchmark.

4.4 Tile Level Site Legalization

Once all of the above steps are run, each tile's density for each type of cell is under control. As pointed out earlier, the BRAMs and REGs are already placed at their best legal location by virtue of the solving assignment problem which maps them to legal locations. However, the LCELLs and DFFs need to be put in their respective columns (see Figure 1). There are two main steps to perform this function:

Column's clock type determination: In the first step for tile level site legalization, we determine which *column* should be reserved for which clock (recall that each column can have cells of only one type of clocks). Our experiments show that this step is very critical since if the column reservation is not done carefully, wirelength can degrade significantly when moving the DFFs to column which can accommodate its clock type. We solve this problem using ILP in a manner similar to solving the problem of deciding which clocks can come in which tile (see Section 4.3). Each column c is assigned as many binary variables as their are clock types. The cost function is the sum of efforts required to clean up a column by removing the DFFs whose clock type is not the same as that returned by ILP solution. A column which initially does not have any DFF in it is assigned a clock number -1 indicating that it can be reserved at a later time while wirelength minimization. Consider a case where upto M DFFs can be accommodated in a column and let each tile have P DFF columns and currently C clocks in it with N_i (i ∈ [1, C]) DFFs for each of these clock types. Binary variable p_i, when true, expresses that column p is reserved for clock i. Let the number of cells before DFF site legalization in the column p of clock type i be given by n_{pi}.

The complete ILP can be written as:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{p=1}^{P}\left(\sum_{i=1}^{C} p_i \times \left(\sum_{j=1, j\neq i}^{C} n_{pj}\right)\right) \\
\text{Subject To :} \quad & \sum_{i=1}^{C} p_i \;\leq\; 1 \qquad\qquad \forall p \in [1, P] \\
& \sum_{p=1}^{P} p_i \;\geq\; \lceil n_{pi}/M \rceil \quad \forall i \in [1, C] \\
& p_i \;\in\; 0, 1
\end{aligned}$$

The ILP above has O(PC) binary variables and O(P + C) constraints. The first set of constraints make sure only one clock can occupy a column. The second set of constraints guarantee enough columns reserved within the tile for each clock type so that its cells can be placed. The values of C and P is determined by the structured ASIC's architecture and is generally a small constant number (e.g. $C = 4$ and $P = 24$ in our benchmarks). In our experiments we observed that the above ILP takes less than 0.2 seconds to solve.

Site Legality of LCELLs and DFFs: With the assignment of all columns to a clock type or unreserved type, we iterate over the LCELLs and DFFs to legalize them. These cells are sorted in non-decreasing horizontal coordinate and are assigned to their closest unoccupied and clock compatible (in case of DFFs) possible location. A procedure *rotateAndPlace* has been implemented which takes as input a given point and a cell and rotates in circle with increasing radius around the point until the given cell can be placed legally depending on the type of the cell.

4.5 Wirelength Recovery

The chip and tile level density legalization outlined in the Section 4.3 and Section 4.4 produce placement results which are completely legal w.r.t. clock, overlap and site constraints. However, due to movement of several cells away from the initial relative order suggested by the placer results on virtual platform, significant increase in the wirelength may occur. Our wirelength optimization procedure depicted in Algorithm 1 recovers wirelength. Our philosophy during wirelength minimization is to never break the clock, site or overlap legality. Algorithm 1 shows the three main phases of our wirelength recovery. In the first phase, large distance inter-tile movement of cells takes place. Using the median idea [15], the best bounding box (BB) of each cell is computed. The cells are then sorted in the non-increasing order of their distance from their best BB and processed in that order. For the cell being currently optimized, a procedure returns the list of all the tiles which have non-empty geometrical intersection with its best BB. This list is iterated

while trying to place the cell near the center of the intersection of best BB and the tile currently being tried using routine *rotateAndPlace*. After this stage, all inter-tile movements have been done and we focus on intra tile wirelength minimization. The second phase improves adjacent columns in the tile to identify any horizontal movement of cells. This is achieved by looking at each consecutive pair of columns such that both are of the same type (DFF or LCELL). Further, if both the columns are DFF columns, their clock type should be same. For such a pair of row, horizontal swapping of a cell in first with the neighboring cell or space in the second column is tried. In the third phase, we look at three consecutive cells/spaces in each column starting from lowest vertical position and enumerate all six possible combinations and pick the best. In all the three phases, we check to see if a movement has caused WL increase and if so, we revert back the change. Our third phase is similar to the method outlined in [16] with the major difference being that we consider white-space also during explicit combination enumeration while FastPlace does not.

Algorithm 1 Wirelength Minimization

Ensure: Placement Is Legal
{ **Large Distance Movement** of Cells}
while WL Improvement $\geq \delta$ **do**
  Compute Best BB b_c for each cell c
  Sort cells according to distance from b_c
  for all cell c in sorted list **do**
    List lst = All tiles intersecting b_c
    for all tile t in list lst **do**
      rotateAndPlace(c, center of t's intersection with b_c)
      if could place in t **then**
        if WL Improved **then**
          break
        else
          continue
{ **Intra Tile Inter Column** WL Improvement}
for all tile t in platform **do**
  while WL Improvement $\geq \delta$ **do**
    for all adjacent columns $p1$ & $p2$ **do**
      if $p1$ and $p2$ are not same type or $p1$ & $p2$ have different clocks **then**
        continue
      for all vertical positions in columns **do**
        Swap cell-cell/cell-space pair adjacent in $p1, p2$
        if WL NOT Improved **then**
          Swap back to original position
{ **Intra Tile Intra Column** WL Improvement}
for all Tile t in platform **do**
  while WL Improvement $\geq \delta$ **do**
    for all Column p in t **do**
      Sort cells and spaces in p by vertical coordinate
      Try all 6 combination of 3 consecutive cells/space
      Pick the best configuration

5. EXPERIMENTAL SETUP AND RESULTS

All the above algorithms are implemented in C++ in our placer tool RegPlace (REGular PLACEr). All our experiments were run on a dual-core 3.3GHz 64-bit AMD linux workstation with 4GB RAM and 8GB swap. We used GLPK [17] as the ILP and network flow solver. The benchmarks were provided by eASIC as part of the placement challenge. The initial placement on the virtual platform were generated using two global placers: mPL [6] and CAPO [5].

Table 2: Benchmark Characteristics

Bench	#LCELL	#DFF	#REG	#BRAM	#CLK
easic1	832,824	87,052	110	172	18
easic2	812,200	45,478	175	686	7
easic3	961,063	52,780	192	0	3
easic4	102,038	23,330	0	44	7
easic5	913,853	84,505	145	262	26

5.1 Advantage of VP Generation

To demonstrate the efficacy of generation of virtual platform (VP) instead of solving the placement problem by blockages, we did the following experiment: Existing row-based placers were run on two benchmarks which were identical to each other except that in one case the placer was run on virtual platform and then mapped back to real platform whereas in the second case, the placer was run by considering blockages. In both the cases, all the cell types were converted to LCELLS, and only the space for LCELLS were available to the placers. Table 3 shows the WL results and time taken to complete the placement for several benchmarks.

Table 3: Advantage of using Virtual Platform (VP) compared to placement with blockages (WB) when using CAPO. Wirelength (WL) are in millions of units. All CPU time are in minutes.

	WL			CPU		
	VP	WB	Δ	VP	WB	Δ
easic1	11.1	11.1	0%	230	464	2.0X
easic2	21.3	21.2	-0.47%	231	476	2.1X
easic3	17.1	17.3	1.15%	374	639	1.7X
easic4	2.0	2.1	5%	29	133	4.5X
easic5	14.55	15.2	4.2%	247	516	2.1X
Average			2.6%			2.5X

Table 3 shows the advantage of using VP to migrate the discrete placement region into a contiguous region compared to the approach of specifying unplacable area as blockages. Overall, by using VP method, the wirelength and runtime improves by 2.6% and 2.5X respectively. On further analysis, we found that most of the extra time is spent by CAPO in overlap removal due to lots of overlaps of cells with the blockages.

While mapping the solution of placement on VP back to real platform, we re-introduce all the blockages. As pointed out in Section 4.2, the wirelength increase due to this step can be reduced by reducing the cut-size by swapping cells lying immediately on the both sides of the introduced blockage. To quantify this benefit, we performed cut-minimization during mapping step on our placement results of Table 3. Only the cells upto two unit distance away on both sides of the blockage re-insertion point were considered during cut minimization to reduce the performance overhead. On an average, we are able to further reduce the WL by 2% over the numbers in previous table.

5.2 Overall Wirelength

A preliminary version of RegPlace competed and won the eASIC placement contest [1]. In its current form, we have further improved the wirelength and runtime of RegPlace by 9% and 10% respectively. Table 4 shows the comparative data for our placer vs. other participants Team1 [18] and Team2 [19]. The WL numbers for Team1, Team2 are as

Table 4: Comparison of our placer with other contestants. CPU numbers for Team 2 unavailable.

Bench	Team1		Team2		RegPlace (M)		RegPlace (C)	
	WL	CPU	WL	CPU	WL	CPU	WL	CPU
easic1	16.9M	3499	10.6M	N.A.	11.3M	13849	12.8M	14538
easic2	32.3M	2444	25.8M	N.A.	23.4M	9529	25.1M	9720
easic3	20.9M	2907	20.5M	N.A.	18.8M	6627	19.6M	7045
easic4	2.3M	216	1.8M	N.A.	2.0M	695	2.1M	735
easic5	20.4M	3055	14.4M	N.A.	13.9M	10197	16.6M	10479
Total	92.8M	12122	73.1M	N.A.	69.4M	40897	76.2	42517

reported in the contest results. However, since we have upgraded our placer, the runtime shown for Team1 and Team2 teams have been scaled by the proportion of runtime our initial version of placer takes on our workstation compared to the contest benchmarking workstations. To understand the impact of using different global placers to generate an initial solution on the virtual platform, we repeated our placer flow with two state-of-the-art placer: mPL and CAPO. The initials (M, C) in last four column names depict use of the above placers respectively.

From Table 4, several observations can be made. Our wirelength results improve significantly (10%) when the initial placement on virtual platform is generated using mPL rather than CAPO. Similarly, the runtime improves on using mPL. Compared to other teams, RegPlace beats Team1 in all the benchmarks irrespective of the initial placer solution being generated with CAPO or mPL. The wirelength reduction w.r.t. Team1 is as much as 33%. However, Team1 does have the advantage of being very fast (3.3X faster than RegPlace). RegPlace when used with mPL for initial placement, is better than Team2 in 3 out of 5 benchmarks as well as the total wirelength. Due to absence of runtime information of Team2, we cannot compare the runtime. The reduction in total wirelength compared to Team2 is 5%. These results show that RegPlace is a high quality solution for structured ASIC placement problem. Figure 4 shows the output of our placer for the benchmark easic2 where each cell type has been plotted with a different color.

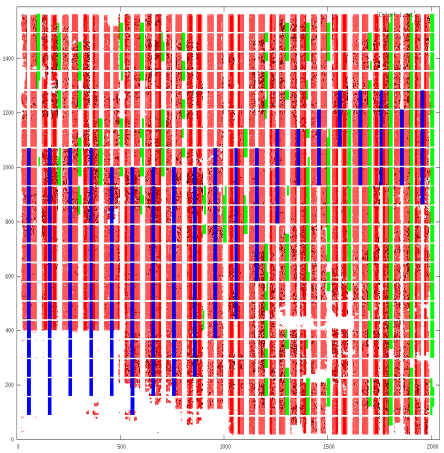

Figure 4: Final layout of benchmark easic2. Cells are shown as LCELL(red), DFF(black), REG(green), and BRAM(blue).

6. CONCLUSIONS

In this work, we proposed a new flow for efficient solution of placement problem for structure ASICs. The key novelty of this work are: a) concept of virtual platform for obtaining better initial placement solution, b) network-flow based inter-tile density correction, c) very fast ILP based clock constraint correction at both chip and tile level. Powered by these techniques, our placer won the eASIC placement challenge. In future, we propose to incorporate multiple density constraints in global placer which can significantly reduce the wirelength penalty associated with site and clock legalization.

7. REFERENCES

[1] "eASIC Corporate Website." `http://www.easic.com/`.

[2] H. Schmit, A. Gupta, and R. Ciobanu, "Placement Challenges for Structured ASICs," in *ISPD '08: Proceedings of the 2008 International Symposium on Physical design*, pp. 84–86, ACM, 2008.

[3] "SOC Central." `http://www.soccentral.com/`.

[4] A. Chakraborty, A. Kumar, and D. Pan, "Regplace," *http://www.cerc.utexas.edu/utda/download/RegPlace.htm*.

[5] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: Robust and Scalable Open-source min-cut Floorplacer," in *ISPD '05: Proceedings of the 2005 International Symposium on Physical design*, pp. 224–226, ACM, 2005.

[6] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mpl6: Enhanced Multilevel Mixed-size Placement," in *ISPD '06: Proceedings of the 2006 International Symposium on Physical design*, pp. 212–214, ACM, 2006.

[7] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion control," in *ASP-DAC '07: Proceedings of the 2007 conference on Asia South Pacific design automation*, pp. 135–140, IEEE Computer Society, 2007.

[8] T. Luo and D. Z. Pan, "DPlace2.0: A Stable and Efficient Analytical Placement Based on Diffusion," in *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pp. 346–351, IEEE Computer Society Press, 2008.

[9] "SOC Encounter tool." `http://www.cadence.com/products/di/soc_encounter`.

[10] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *FPL '97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp. 213–222, Springer-Verlag, 1997.

[11] K. Eguro, S. Hauck, and A. Sharma, "Architecture-adaptive Range Limit Windowing for Simulated Annealing FPGA Placement," in *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pp. 439–444, ACM, 2005.

[12] A. C. Ling, D. P. Singh, and S. D. Brown, "Incremental Placement for Structured ASICs Using the Transportation Problem," *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, pp. 172–177, Oct. 2007.

[13] T. Okamoto, T. Kimoto, and N. Maeda, "Design Methodology and Tools for NEC Electronics' Structured ASIC ISSP," in *ISPD '04: Proceedings of the 2004 International Symposium on Physical design*, pp. 90–96, ACM, 2004.

[14] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[15] S. Goto, "An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit layout," *Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 12–18, Jan 1981.

[16] M. Pan, N. Viswanathan, and C. Chu, "An Efficient and Effective Detailed Placement Algorithm," *Computer-Aided Design, International Conference on*, vol. 0, pp. 48–55, 2005.

[17] "GNU Linear Programming Kit." `http://www.gnu.org/software/glpk/`.

[18] "Michigan Wolverine Placer by Dongjin Lee and Igor Markov."

[19] "The Esteem Placer, written by Bob Erickson, an independent software consultant." `http://www.linkedin.com/in/boberickson`.