Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells

Gang Wu and Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, IA
Email: {gangwu, cnchu}@iastate.edu

***Abstract*—Conventional detailed placement algorithms typically assume all standard cells in the design have the same height. However, as the complexity and design requirement increase in modern VLSI design, designs with mixed single-row height and double-row height standard cells come into existence in order to address the emerging standard cell design challenges. A detailed placement algorithm without considering these double-row height cells will either have to deal with a lot of movable macros or waste a significant amount of placement area, depending on what type of techniques people use to accommodate such design. This paper proposes a new placement approach which can handle designs with any number of double-row height standard cells. We transform design with mixed-height standard cells into one which only contains same height standard cells by pairing up single-row height cells into double-row height. Then conventional detailed placement algorithms can be applied. In particular, we generate cell pair candidates by formulating a maximum weighted matching problem. A subset of the cell pair candidates are then carefully selected to form double-row height cells based on the local bin density. A refinement procedure is performed at the end to further improve our placement quality. We compare our approach with two alternative detailed placement methods on mixed-height asynchronous and synchronous designs. The experimental results show that our approach can achieve much better quality and robustness.**

I. Introduction

Standard cell methodology has been widely adopted as a quick and efficient method to overcome the continuously increasing complexity of integrated circuit design. In standard cell library design, cell height is fixed to an integer multiples of a unit row height, but cell width can be variable. Conventionally, a standard cell library only contains cells with single-row height [1], as smaller cell height can achieve a higher density for simple standard cells (e.g., inverter, nand, nor) and hence lower the cost. However, for complex standard cells (e.g., flip-flop, latch), the limitation on cell height will create heavy routing congestion. In this case, the persistence in single-row height standard cells will greatly decrease layout efficiency and consume more layout design time from engineers [2]. Also, for high performance applications, single-row height cells might not be able to deliver sufficient current because the transistor size is small [3]. Thus, multi-row height standard cells, in particular double-row height cells, are commonly intermixed with smaller single-row height standard cells in order to increase the area, design efficiency and help meet the design requirements [4][5].

Another case which motivates us to pay attention to double-row height cells is their common existences in some other VLSI design styles. For example, for asynchronous circuit design, most of its standard cell can be double-row height. This is because asynchronous logics require extra circuits to generate handshaking signals [6][7], which means a larger and more complex cell is required compared with their synchronous counterparts.

This work is supported in part by NSF award CCF-1219100.

Placement has become a very critical step in today's VLSI physical design flow. While double-row height placement is available in commercial tools, it is still new in academic field and most placement techniques typically assume all single-row height cells to be standard cells and other cells to be fixed or movable macros [8]. The detailed placement algorithm will focus more on improving the placement of standard cells while relying on floorplanning techniques to place macros into a good location [9][10]. If the design only has a very small amount of double-row height cells, treating them as macros works well. However, there are designs which have more double-row height cells. In our case, the set of asynchronous benchmarks we used for our experiment has an average of 77% double-row height standard cells in each design. If we treat all double-row height cells here as movable macros, even the initial legalization step can take a lot of runtime and we might still get overlaps in the end. This is because the floorplanning techniques will not be scalable to hundreds and thousands of movable macros [11]. Also, during later detailed placement steps, no optimization (such as the commonly used cell swapping methods [9][12]) is applied to these cells, as their locations are fixed in the very beginning. Therefore, the placement quality of the design will be greatly affected.

An alternative way to place these design is to expand all single-row height cells to double-row height. Then all standard cells will have the same height which is compatible to conventional detailed placement algorithms. However, the cell expansion can increase chip utilization quite a lot, depending on how many single-row height cells we have in our design. If the chip utilization becomes too high, there will not be enough free space for the detailed placement algorithm to explore a good placement solution. If the utilization becomes more than 100%, we will even not be able to obtain a legalized placement. Also, cell expansion can lead to some cells not be able to be placed closer together which means the wirelength will be increased.

In this paper, we are focusing on the problem of detailed placement for designs with any number of double-row height cells. The input to our placement problem is a placement region, a set of modules, and a set of nets. Also, we are given a set of rough locations for each modules which is obtained from the global placement result. Our algorithm finds a position for each module within the placement region so that there is no overlap among the modules and the total wire length is minimized. Our idea is to transform mixed-height cells in a design into the same height. Then conventional detailed placement techniques can be applied on them. The equalization of cell heights is realized using a combination of two techniques: cell expansion and cell pairing. In particular, in low density areas, we apply cell expansion by doubling the height of single-row height cells to have minimum restriction on the cell movement. While in high density areas, appropriate pairs of single-row height cells are identified and combined into double-row height cells to achieve more available

free space compared with simply doing cell expansion.

Our detailed placement approach is compared with two alternative placement methods: for the first method, all the height of single-row height cells will be doubled before doing detailed placement. Then conventional detailed placement algorithm is applied on this equal cell height but expanded design. For the second method, we directly applied conventional detailed placement algorithm, which means all double-row height cells will be treated as movable macros during the detailed placement process. The experimental results show that our placement approach can always achieve a better wirelength compared with the other two methods and is much more robust in handling designs with different amount of double-row height cells.

The rest of the paper is organized as follows: Section II provides an overview of our approach. Section III explains each technique we used in details. Section IV shows the experimental results compared with other methods. Finally, Section V concludes the paper.

II. Overview

Our goal here is to develop an algorithm which can handle designs with any number of double-row height cells, while minimize the total wirelength of the design. Although we only consider wirelength in this paper, other objectives (e.g., timing, routability, manufacturability) can easily be considered by using a detailed placer optimizing those objectives.

A single-row height cell can be changed to double-row height by either simply expanding the cell or pairing up two single-row height cells together and form a double-row height cell. Pairing up cells can help to place them more tightly, which reduces the local bin density and provide more free space for detailed placement algorithm to explore a good solution. However, at the same time, forcing two cells to be placed together will restrict their movement and some potential placement solution cannot be explored. In order to make sure that the formed cell pairs do not impose too much restriction during the detailed placement, we only pick those pairs which can provide us the most "benefit", which will be defined in Sec. III. Also, we go back to the simple cell expansion strategy in low density areas where there is sufficient free space even after expansion, in order to give each single-row height cell the maximum freedom to move.

A high-level view of the flow developed in this paper is shown in Fig. 1.

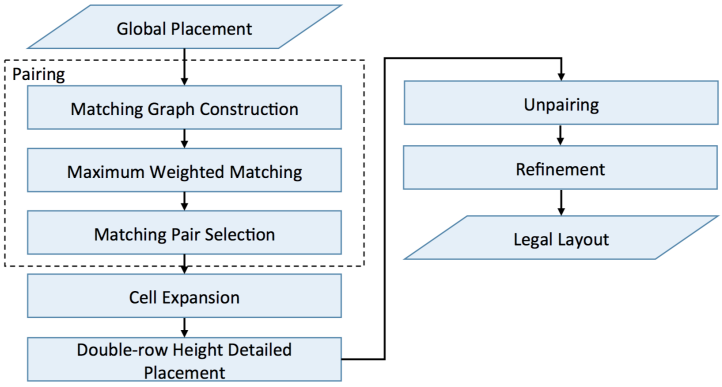

Fig. 1: Detailed Placement Flow

Our flow works on a global placement result. The flow starts with a cell pairing procedure, which is formulated as a maximum weighted matching problem and composed of three stages: matching graph construction, maximum weighted matching and matching pair selection. Each pair of single-row height cells will be merged into a double-row height cell. In cell expansion step, single-row height cells which have not been paired up will be expanded to double-row height. Next, conventional detailed placement algorithm is applied on the transformed design which only contains double-row height cells. At the end, a refinement procedure is performed based on the previous detailed placement results, in which we fix the location of double-row height cells and run detailed placement algorithm on unpaired single-row height cells in order to further improve the placement quality.

III. Detailed Placement Approach

A. Matching Graph Construction

The benefit of pairing up single-row height cells is modeled using a matching graph here. The first thing we need to consider while constructing the matching graph is a good trade off between solution space and algorithm running time. If we simply construct a matching graph which an edge exists between any two single-row height cells in the design, we are able to explore all possible cell pairing solutions. However, we will end up having a complete graph with the number of edges quadratic to the number of single-row height cells and the runtime of our matching algorithm will not be acceptable. On the other hand, if we construct a matching graph which each cell is only connected with very few other cells, the total number of pairing candidates for a cell will be too small. It is quite possible that some good cell pairs are missed.

Our matching graph is constructed like this: We divide the placement region into $m \times n$ equal sized bins. Only cells within neighboring bins are considered to help limit the choice when we search for candidates. In particular, for each single-row height cell u, we first locate the bin containing this cell. Then we look for all other single-row height cells $V = \{v_1, ..., v_n\}$ within this bin and the nearest neighboring bins. Next, we want to create an edge between u and any $v_i \in V$, if the Manhattan distance between u and v_i is less than a target distance r. Here the target distance r is carefully selected such that each cell will have enough candidates to be chosen from and the overall running time of the matching algorithm will not be too much. Ignoring cells out of range r to be a candidate does not make a big impact on the quality of matching algorithm, as pairing up cells far away can dramatically change the global placement result and make the overall wirelength worse.

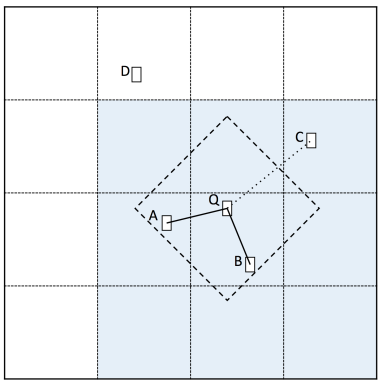

Fig. 2: Construct Matching Graph

An example of matching graph construction is shown in Fig. 2. To search for matching candidates of cell Q, we first look at all single-row height cells within the shaded region, which is cell A, B and C. Cell D will be ignored in this case. Then we check if the distance between (Q, A), (Q, B) and (Q, C) is within a feasible range shown as a doted diamond in this figure. In this example, A and B is selected as matching candidates and edges (Q, A) and (Q, B) are added to the matching graph. We do not consider C as a matching candidate, as C is out of the range.

B. Edge Weight Calculation

We want to calculate edge weight based on the associated benefit if two cells are paired up. Three different factors are considered during our weight calculation: cell connectivity, area increase and cell displacement.

1) Cell connectivity: Here we want to consider the connectivity between cells and give more chance to those with strong connectivity in the netlist to form a pair.

Given two single-row height cells u and v. Let C be the connectivity factor for edge (u, v) in our matching graph. Let e be a hyperedge connecting cells u and v in the netlist. Let $|e|$ be the number of cells that are incident to this hyperedge. Clique model is applied here to decompose hyperedges.

We define C as:

$$C = \sum_{e \in E | u,v \in e} \frac{1}{|e|}$$

where E is the set of hyperedges in the netlist.

2) Penalty on area increase: The width difference between two cells can play an important role while we form pairs. Consider a simple example which four cells A, B, C and D want to form into two pairs. If a wide cell is paired up with a thin cell as shown in Fig. 3 (a), a lot of placement area will be wasted after grouping this two cells together. Instead, if we pair up cells with similar width as shown in Fig. 3 (b), the total area after pairing will be much smaller than the previous method.

(a) (b)

Fig. 3: (a) Pair up cells with large width difference (b) Pair up cells with small width difference

Here we define PA as the penalty on area increase. Consider two standard cells u and v which have single-row height h. We can set PA as:

$$PA = h * |W_u - W_v|$$

where W_u and W_v are the widths of cells u and v.

3) Penalty on cell displacement: Another factor we considered is cell displacement. We want to encourage pairing up two cells which are closer together while adding penalty on forming pairs which the two cells are relatively far away. The idea is to minimize the perturbation of global solution, such that the total wirelength will not be affected too much after pairing up cells.

Let PD be the penalty of cell displacement. Let $d(u, v)$ be the Manhattan distance between two cells u and v. The cell displacement penalty can be defined as:

$$PD = d(u, v)$$

Putting everything together and let B be the benefit of forming a pair, we can get:

$$B = C - \alpha_1 * PA - \alpha_2 * PD$$

where constant α_1 and α_2 can be chosen to adjust the effect among cell connectivity, area penalty and displacement penalty.

C. Maximum Weighted Matching

We use maximum weighted matching to find a set of candidate cell pairs such that pairing them up will result in maximum benefit. Matching problem is one of the most well-studied problems in computer science. There are many existing algorithms in the literature which can either find a perfect matching or do some approximation [13]. In order to find the one which is most suitable to our problem, we implemented two different matching algorithms in our experiment to perform weighted matching. One is based on the Edmond's maximum weighted matching algorithm [14] which provides $O(nmlog(n))$ runtime complexity and another is a simple greedy approach [15] which can find an approximate solution within a linear runtime. The experiment results show that these two approaches provide similar wirelength results, while the second approach has a much better runtime. Thus, the second one is chosen in our flow.

D. Matching Pair Selection

It is not the best to pair up all single-row height cells from the matching result. Forming a pair will force the two cell to be placed together and put restrictions on detailed placement. For the matched pairs with small edge weight, the benefit they provide cannot justify to tie them up. This is especially true for matched pairs in low density area. It would be better to simply expand those cells and give them freedom to move, since in low density area, there will always be enough free space even after cell expansion.

In our flow, we only pick the top portion of matching results based on edge weight and local bin density to perform actual cell pairing, while the remaining single-row height cells are simply expanded to become double-row height.

We should notice that, after the cell pairing and expansion, the local bin density might change. The new density will depend on not only the initial bin density of the given placement, but also the total area of single-row height cells which do not form a pair. This is because the unpaired single-row height cells will double their area when they are expanded to double-row height.

Here we divide the chip into $p \times q$ equal sized bins. The size of the bins we used here can be different from the one we used during matching graph construction in Sec. III A. Let D_b be the density for bin b after global placement. Let k be the area percentage of single-row height cells which do not form a pair in this bin. Then the new density D'_b after pairing and expansion will be: $D'_b = 2 * kD_b + (1 - k)D_b = (1 + k)D_b$. Here D'_b is just an approximate value. Pairing process might move single-row height cells from one bin to another bin which can affect the bin density. There will also be some area wastage after we pack two single-row height cells together. However, considering both the cell movement and area wastage is small, we simply ignore these facts to make our algorithm simple. As we are only locally

estimating D'_b, for designs with very high utilization, there might be an utilization overflow issue after cell expansion. But because of what we did, the chance of having such an issue is very unlikely and we have never encountered any issue in practice.

After cell candidates are generated, we first globally filter out matched pairs which have a small edge weight. Then for each bin, we select part of the matching results to form cell pairs based on D'_b. In particular, we want to keep D'_b within a good range to make sure that each bin will have enough free space and we also do not form too many pairs to limit the cell movement. The selected single-row height cell pairs are then paired up into double-row height cells with the new location in the middle of the corresponding single-row height cells.

E. Unpair and Refinement

After we run conventional detailed placement algorithm on the transformed design and get a legalized solution, there might still be room for wirelength improvement. First is because the pairs we formed in the design restrict two cells not be able to be placed in separate places. Second, the expansion on some single-row height cells also limits them to be put closer to other cells. Thus, we run detailed placement for a second time without these restrictions by unpairing cells and deflating the expanded cells. The locations of double-row height cells are fixed at this run, as we assume they have already been placed into a good location during the previous detailed placement process.

IV. Experimental Results

The proposed approach is implemented using C++ and run on a Linux PC with 8 GB of memory and Intel 2.4 GHz CPU.

Two sets of benchmarks are used in our experiment. First is a set of asynchronous VLSI designs synthesized using an asynchronous frond-end flow [16]. Another set of benchmarks are created based on the ISPD05 placement benchmark suite. We randomly selected about 30% single-row height standard cells in the design and doubled their height. The placement region area is keep to be the same. POLAR [17] is used to generate global placement results as an input to our flow.

We choose FastDP [9] as our detailed placement engine with small modification to make sure double-row height cells are placed only on even rows. Since FastDP is designed for single-row height benchmarks, if all standard cells are placed on even rows while some macros are aligned with odd rows, FastDP might create some overlaps. We solve this problem by adding placement blockages on the row above and / or below each macros. Two alternative methods are developed to be compared with our approach. In method 1, we apply cell expansion technique in which we double the height of single-row hight cells. Then, FastDP is run on this expanded design with all cells having equal height. In method 2, we treat the double-row height cells as movable macros. The design is first legalized using techniques described in [18], then we use FastDP to perform the detailed placement. The values of α_1 and α_2 are experimentally determined and we set $\alpha_1 = 2 \times 10^{-3}$ and $\alpha_2 = 2 \times 10^{-4}$ for all the benchmarks.

Comparison results on asynchronous benchmarks are shown in Table I. Second column shows the total number of cells. The percentage of double-row height cells and the chip utilization are shown in the third and fourth column. The "Init" column shows the wirelength of the global placement results after legalization. For the wirelength, our approach is 3% better than the first method and 14.8% better than the second method. The runtime of our approach is a little bit worse than the first method, as our approach run FastDP twice, but we are much better compared with the second method. The "Overlaps" column shows the number of overlaps reported by FastDP after the detailed placement. Both our approach and method 1 can generate a legalized placement with zero overlaps. However, for the second method, almost all the designs cannot be legalized, as there are too many macros which is beyond the ability of FastDP to handle.

Table II shows the comparison results on synchronous benchmarks. On average, our wirelength is 72.2% better than the first method and 3.5% better than the second one. Also, our algorithm runtime on average is better compared with the other two methods. For method 1, the reason is because the cell expansion increases chip utilization too much and ends up making FastDP take a much longer time to perform optimization. For method 2, its runtime is better than our approach only when the design size is small. As the design size increases, legalization techniques used inside FastDP will not be scalable to the increasing number of movable macros. Thus, the runtime becomes much slower. We can also see that method 1 failed to place some designs, as their chip utilization become greater than 1 after we expanded single-row height cells in this two designs.

To further illustrate the robustness of our approach, we did another set of experiments using adaptec2 and bigblue1. We generate a new set of benchmarks with different percentage of double-row height cells in a design by randomly picking certain number of single-row height cells and doubling their height. Then, our approach and the other two methods are run on this new set of benchmarks. The original adaptec2 and bigblue1 design have the utilization of 81% and 61%. Since most of the cell area is occupied by macros, there will not be an utilization overflow.

The results are shown in Fig. 4 and Fig. 5. X axis is the ratio of double-row height cells in the design and Y axis is the total wirelength. It can be seen that method 1 which apply cell expansion techniques do a very bad job when single-row height cells are dominating in the design. The total wirelength gradually get improved with the increasing number of double-row height cells. In contrast, method 2 which treats all double-row height cells as macros works well when double-row height cells are not so many, but the wirelength gets worse when total number of double-row height gets increased. For adaptec2_dr, method 2 cannot even finish within a reasonable amount of runtime when the number of double-row height cells below 30%. In comparison, no matter how many double-row height cells exists in the design, our approach always produces placement results with better wirelength.

V. Conclusions and Future Work

In this paper, we have proposed a detailed placement approach targeting at designs with mixed single-row height and double-row height standard cells. We incorporated cell paring and cell expansion techniques and transformed the mixed-height design into design containing only standard cells of the same height. Then any conventional detailed placement algorithm can be applied. Our approach is compared with other two alternative methods to place design with mixed-height standard cells and achieves both better quality and robustness.

Our future work is to incorporate bin utilization constraints into our algorithm. In particular, we want to consider bin utilization

Table I. Comparison on asynchronous benchmarks

Design	Size	DH Cells	Util	Wirelength x 10^6 (nm)				Runtime (s)			Overlaps		
				Init	Ours	Method1	Method2	Ours	Method1	Method2	Ours	Method1	Method2
s9234	2108	85.10%	75.02%	167.74	134.10	133.55	159.04	1.14	1.21	244.36	0	0	294
s15850	6778	81.45%	75.35%	671.51	561.80	575.36	650.43	3.09	2.55	2112.03	0	0	1372
s13207	5658	84.36%	77.73%	521.02	432.58	435.20	509.90	2.72	2.97	1729.70	0	0	1104
s38417	15447	61.68%	68.73%	1457.15	1176.30	1223.99	1331.89	5.99	4.77	6211.27	0	0	200
ALUMAN16	2442	61.38%	63.26%	230.69	174.16	181.71	196.78	3.99	2.72	171.86	0	0	6
ALUMAN32	6866	63.17%	66.22%	819.71	643.14	692.35	733.20	3.84	5.49	1470.53	0	0	36
ALUMAN64	28974	70.69%	70.59%	3863.54	3204.05	3310.77	3659.70	15.18	9.05	26576.30	0	0	2018
acc64	3355	92.28%	75.68%	275.48	225.39	224.90	269.17	3.67	2.04	605.15	0	0	800
acc128	8401	90.44%	79.05%	692.96	601.34	604.00	682.36	8.90	6.32	3239.38	0	0	2760
GCD	445	67.64%	63.30%	36.35	24.14	25.09	30.00	1.00	0.45	6.39	0	0	0
FetchingUnit	5304	92.18%	75.05%	674.84	563.58	566.45	667.37	3.63	1.98	1411.80	0	0	1350
				1.216	1	1.030	1.148	1	0.744	823.612			

Table II. Comparison on synchronous benchmarks

Design	Size	DH Cells	Util	Wirelength x 10^6 (nm)				Runtime (s)			Overlaps		
				Init	Ours	Method1	Method2	Ours	Method1	Method2	Ours	Method1	Method2
adaptec1_dr	211447	30.18%	90.84%	97.24	91.35	Fail	94.28	38.33	Fail	28.49	0	Fail	0
adaptec2_dr	255023	30.16%	89.12%	109.07	105.66	242.12	107.94	44.36	300.86	42.62	0	0	0
adaptec3_dr	451650	30.11%	78.44%	249.69	242.13	433.40	245.89	82.44	417.53	75.98	0	0	0
adaptec4_dr	496045	30.19%	67.70%	214.30	208.92	295.04	211.46	84.03	296.20	65.83	0	0	0
bigblue1_dr	278164	30.14%	73.44%	117.73	113.09	181.99	115.42	41.60	244.19	38.47	0	0	0
bigblue2_dr	557866	32.90%	68.99%	169.32	160.86	278.03	164.62	85.88	262.05	87.33	0	0	0
bigblue3_dr	1096812	30.31%	91.10%	490.91	418.97	Fail	466.60	233.67	Fail	440.98	0	Fail	423
bigblue4_dr	2177353	30.26%	73.88%	913.23	882.51	Fail	895.39	469.78	Fail	753.53	0	Fail	171
				1.062	1	1.722	1.035	1	4.495	1.420			

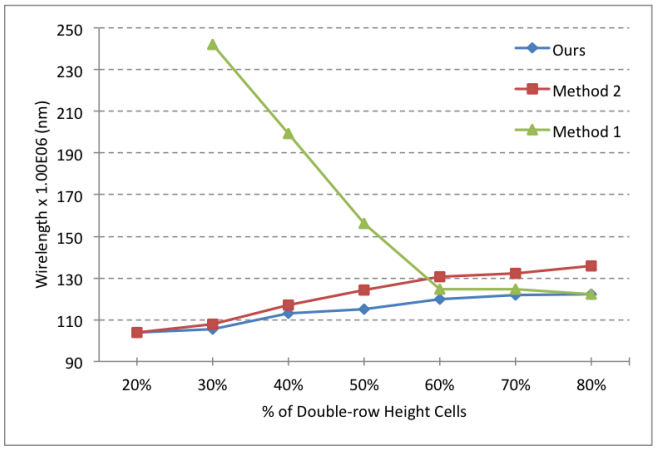

Fig. 4: Experiment on adaptec2_dr benchmark

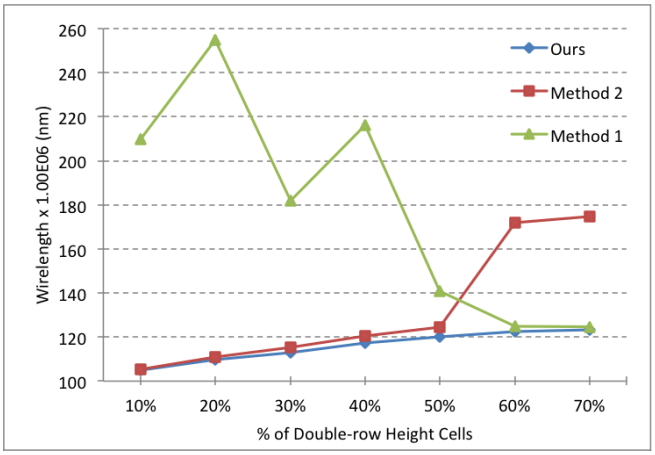

Fig. 5: Experiment on bigblue1_dr benchmark

during the edge weight calculation in Sec III-B and use a detailed placement engine which supports bin utilization constraints.

REFERENCES

[1] J. Wang, A. K. Wong, and E. Y. Lam, "Standard Cell Layout with Regular Contact Placement," *Semiconductor Manufacturing*, 2004.

[2] S. H. Baek, H. Y. Kim, Y. K. Lee, D. Y. Jin, S. C. Park, and J. D. Cho, "Ultra-High Density Standard Cell Library Using Multi-Height Cell Structure," in *Smart Materials, Nano-and Micro-Smart Systems*, 2008.

[3] D. G. Breid, M. J. Colwell, T. R. Gheewala, and H. H. Yang, "Dual-Height Cell with Variable Width Power Rail Architecture," US Patent, 2005.

[4] A. P. Hurst, P. Chong, and A. Kuehlmann, "Physical Placement Driven by Sequential Timing Analysis," in *ICCAD 2004*, pp. 379–386, Nov 2004.

[5] S. Dobre, A. B. Kahng, and J. Li, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes," in *ICCAD 2015*, pp. 854–860, IEEE Press, 2015.

[6] A. M. Lines, *Pipelined Asynchronous Circuits*. Master's thesis, California Institute of Technology, 1998.

[7] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[8] M. C. Kim, N. Viswanathan, Z. Li, and C. Alpert, "ICCAD-2013 CAD Contest in Placement Finishing and Benchmark Suite," in *ICCAD 2013*.

[9] P. Min, N. Viswanathan, and C. Chu, "An Efficient and Effective Detailed Placement Algorithm," in *ICCAD 2005*, pp. 48–55, Nov 2005.

[10] S. Popovych, H. H. Lai, C. M. Wang, Y. L. Li, W. H. Liu, and T. C. Wang, "Density-Aware Detailed Placement with Instant Legalization," in *DAC 2014*, pp. 1–6, June 2014.

[11] S. N. Adya and I. L. Markov, "Combinatorial Techniques for Mixed-Size Placement," *TODAES*, vol. 10, no. 1, pp. 58–90, 2005.

[12] J. Cong and M. Xie, "A Robust Detailed Placement for Mixed-Size Ic Designs," in *ASP-DAC*, p. 7 pp., 2006.

[13] R. Duan and S. Pettie, "Linear-Time Approximation for Maximum Weight Matching," *Journal of the ACM (JACM)*, vol. 61, no. 1, p. 1, 2014.

[14] J. Edmonds, "Maximum Matching and a Polyhedron with 0, L-Vertices," *Journal of Research of the National Bureau of Standards*, 1965.

[15] D. E. Drake and S. Hougardy, "A Simple Approximation Algorithm for the Weighted Matching Problem," *Information Processing Letters*, 2003.

[16] P. A. Beerel, G. Dimou, and A. Lines, "Proteus: An ASIC Flow for GHz Asynchronous Designs," *Design Test of Computers*, 2011.

[17] T. Lin, C. Chu, J. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: Placement Based on Novel Rough Legalization and Refinement," in *ICCAD 2013*, pp. 357–362, Nov 2013.

[18] N. Viswanathan, M. Pan, and C. C.-N. Chu, "FastPlace: an analytical placer for mixed-mode designs," in *ISPD*, pp. 221–223, 2005.