Quantifying Academic Placer Performance on Custom Designs

Samuel Ward
IBM STG
11400 Burnet RD
Austin TX 78758
siward {@us.ibm.com}

David A. Papa
IBM Austin Research
11501 Burnet Rd.
Austin, TX 78758
iamyou {@eecs.umich.edu}

Zhuo Li
IBM Austin Research
11501 Burnet RD
Austin TX 78758
lizhuo{@us.ibm.com}

Cliff Sze
IBM Austin Research
11501 Burnet RD
Austin TX 78758
csze {@us.ibm.com}

Charles Alpert
IBM Austin Research
11501 BURNET RD
AUSTIN TX 78758
alpert {@us.ibm.com}

Earl Swartzlander
The University of Texas at Austin
Electrical and Computer Engineering
Austin, TX 78712 USA
eswartzla {@aol.com}

ABSTRACT
There have been significant prior efforts to quantify performance of academic placement algorithms, primarily by creating artificial test cases that attempt to mimic real designs, such as the PEKO benchmark containing known optimas [5]. The idea was to create benchmarks with a known optimal solution and then measure how far existing placers were from the known optimal. Since the benchmarks do not necessarily correspond to properties of real VLSI netlists, the conclusions were met with some skepticism. This work presents two custom constructed datapath designs that perform common logic functions with hand-designed layouts for each. The new generation of academic placers is then compared against them to see how the placers performed for these design styles. Experiments show that all academic placers have wirelengths significantly greater then the manual solution. These testcases will be released publically to stimulate research into automatically solving structured datapath placement problems.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids – Placement and Routing

General Terms
Algorithms, Design, Experimentation

Keywords
Standard Cell Placement, Datapath Placement, Placement Benchmarks

1 INTRODUCTION
Automatic VLSI placement algorithms have improved significantly since the ISPD placement contests in 2005 and 2006 [14] [15]. These contests released 16 new placement benchmarks derived from industrial designs.

The benchmarks contained a number of important features that were not present in the previous set of benchmarks: (i) they ranged in size from 211k cells to 2.18M cells, much larger than previous benchmarks (ii) they contained large fixed obstacles not seen in previous benchmarks (iii) they contained large movable objects which cover more than a single circuit row in height, a feature that was added to existing benchmarks [1].

In addition, the structure of the contest forced placement algorithms to optimize half-perimeter wirelength (HPWL), runtime and a target density, which is used in practice to improve both timing and routability of circuits in physical synthesis. Prior to the contests, few academic placers could solve these realistic problem instances, though that is certainly not true today [3] [4] [10] [16]. It is easy to observe that benchmarks are important to guide the development of practical placement algorithms.

Prior to these contests, there were attempts to quantify the suboptimality of placement heuristics. Hagen, *et al.* [7] had the idea of taking copies of small circuits and replicating them, then loosely connecting their ports together, in order to create a much larger benchmark. For example, by connecting four copies of a well-placed circuit together in 2 x 2 grid, they obtained a placement wirelength that was no more than four times that of the original circuit. While interesting, this experiment is arguably unrealistic since these defined connections between the copies do not correspond to real logic functions. Furthermore, no pin locations are defined for the circuit (nor were there any for the original). This work overcomes both prior objections.

More recently, Chang, *et al.* [5] created the placement examples with known optima (PEKO) and placement examples with known upperbounds (PEKU) algorithms and released two sets of benchmarks with solutions that are known to be optimal or close to optimal. Optimality was achieved by adding nets to cells in configurations that cannot be shortened. In other words, they created a design where every net was a super-short net, though the pin distributions of cells matched that of a typical VLSI circuit. Reported results show wirelengths in the range of 1.43 to 2.40 times the optimal value. Again while interesting, these netlists did not correspond to any logic function at all. It could be argued

that the PEKO and PEKU testcases are artificially hard and that no placer would ever need to solve them.

Given the renaissance in automated placement technology that has occurred, it seems like a good time to revisit this issue of quantifying placement algorithms. Perhaps this new generation is close to optimal, especially given that placer improvements are worthy of publication when they manage to obtain a 1-2% improvement in wirelength. However, unlike previous efforts, this work quantifies placement algorithms on useful logic function. It is accepted folklore that current placement tools do not perform particularly well on custom or structured designs. Due to their regular structure, datapath designs enable a designer to construct highly compact custom layouts. To shed light on this issue, the solutions of academic placement tools are compared on two manual designs created for this purpose.

The initial design for each circuit was developed using standard, custom design practices. Logic gates from automated design standard cell libraries were hierarchically built within a custom schematic design framework. Each design used a reduced library of basic 2-, 3-, and 4-input NAND, NOR, INVERT, MUX and XOR gates and latches.. The combinational logic gates for both benchmarks were allowed to move during the course of automatic placement by several academic tools [19].

Many have speculated that poor performance of placers on datapath designs is due to very tight density constraints. Perhaps placers could find the right structures but simply had trouble with the legalization. Consequently, eight variants of each design were created where additional whitespace was inserted to provide more opportunity for the placers. While wirelength was improved, all placers still generated solutions with wirelengths at least 1.12 times that of the custom solution. The empirical results confirm there remains significant room for improvement in modern academic placement algorithms.

The paper is organized as follows. Section 2 presents a rotate circuit, and shows a common manual layout solution. Section 3 does the same for a compare logic circuit. Experiments results comparing six placers are presented in Section 4. Conclusions are presented in Section 5.

2 DESIGN 1: ROTATE LOGIC

Rotate circuits, also known as cyclic shifters [8] [11] [12], are a simple and common bit operation generally found throughout microprocessors, cryptography, imaging, and biometrics [2] [13]. Traditionally, rotators are custom designed because of their highly regular structure and significant routing complexity [6] [18] though some work on automated placement has been explored [9].

2.1 Overview

A standard rotate function consists of cascaded 2-input MUXes, as shown in Figure 1. A rotator circuit receives a set of inputs $d[0:n-1]$ and $r[0:m-1]$ and produces an output $s[0:n-1]$, where $d[0:n-1]$ has been rotated by some amount encoded by $r[0:m-1]$. In the following notation, & indicates a logical AND, + indicates a logical OR, and ! indicates a logical NOT. To mathematically define the rotate functions, let $k[i,j]$ denote the internal point at ith row and jth column in Figure 1, where $i = (0:m-1)$ for $r[0]$ to $r[m-1]$ and $j = (0:n-1)$ for $d[0]$ to $d[n-1]$. Then, $k[0,j] = !\,(\,r[0]\,)\,\&\,d[i] + r[0]\,\&\,d[i+1]$, where j = 0, …, n-1 and note that $n = 2^m$. Thus the general equations are:

$$k[i,j] = !\,(\,r[i]\,)\,\&\,k[i-1\,,\,j] + r[i]\,\&\,k[\,i-1\,,\,j+2^i]$$
where i = 0, ..., m-1 , j = 1, ..., n-1

$$k[i,\,j] = k[i,\,j + z * n], \quad \text{where } z \text{ is } 0, 1, 2, \ldots,$$

Figure 2 shows an example of an eight-way rotate function. The initial input vector $d[0:7]$ = { 01110101 } and $r[0:2]$ = { 101 }, indicating a rotation of five. In the first stage, $r[0]$ rotates the input vector one bit position, $r[1]$ in stage two does not rotate the vector, and in the third stage, $r[2]$ rotates the vector four more bit positions for a result of $s[0:7]$ = { 10101110 }.

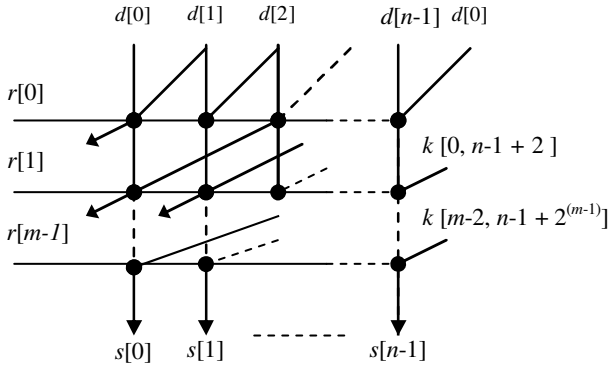


Figure 1. Rotate Block Diagram

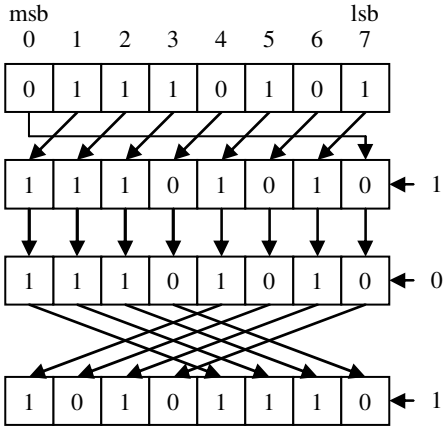


Figure 2. Rotate Example

Rotator designs present automated design tools with the challenge of producing a densely-packed placement solution while minimizing routing congestion. There are two parts to the routing challenge, local routing and global routing. Local routes between each MUX must be lined up very carefully to leave space for the global select lines $r[0:m-1]$. At each stage, the route from the previous stage shifts one more column

over, creating a congested routing network. Design placement that minimizes jogging global routes is critical to achieve a routable design that meets area and timing constraints. In addition, careful attention to the design of global routes is necessary for optimal delay.

2.2 Benchmark Details

The first benchmark in this paper, Design 1, derives from the manual placement of an actual high-speed microprocessor rotate function. The logic implementation also includes two enable signals at each rotate circuit. Certain portions of the design are modified, such as the intermediate output pins and the latch points, without modifying overall functionality. Figure 3 displays the basic MUX and the enable building block for the design, which is referred to as a complex subcell. Each complex subcell is comprised of a two-to-one MUX with a corresponding select signal $r[i]$ and enable signals $e_h[i]$ and $e_v[j]$. Enable signal $e_h[i]$ runs horizontally to each bit stack in the ith row, and $e_v[j]$ runs vertically to each complex subcell within a bit stack at jth column. Exact circuit implementation can vary, depending on the specific technology; however, in this design, a single two-to-one MUX and a three-input NAND gate are used for the implementation. Each following stage is inverted to maintain polarity without impacting TWL calculations.

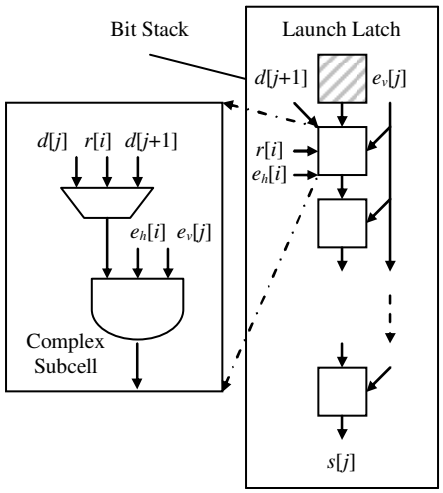


Figure 3. Rotate Sub-block

Using the notation from Figure 1, Design 1 contains n=511 and m=63, which means it is a 512 bit rotate circuit with 9 encoding bits. Each $d[0{:}n\text{-}1]$ is stored in a latch with a fixed location and drives the stacked MUX structure, which is nine complex subcells high. Primary input (PIs) and output pins (POs) were placed directly on top of their respective connections minimizing PI/PO routing distance.

2.3 Placement Details

Figure 4 shows a representative layout for an 8-bit rotator as in Figure 2. The data bus $d[0{:}7]$ initially resides in the latches denoted *lat* with each complex subcell stacked directly on top. The *mux* is the 2:1 MUX in the complex subcell and *and* is the AND-3 gate in the complex subcell. The rotate result, $s[0{:}7]$ leaves the top of each bit stack driven from the last complex subcell.

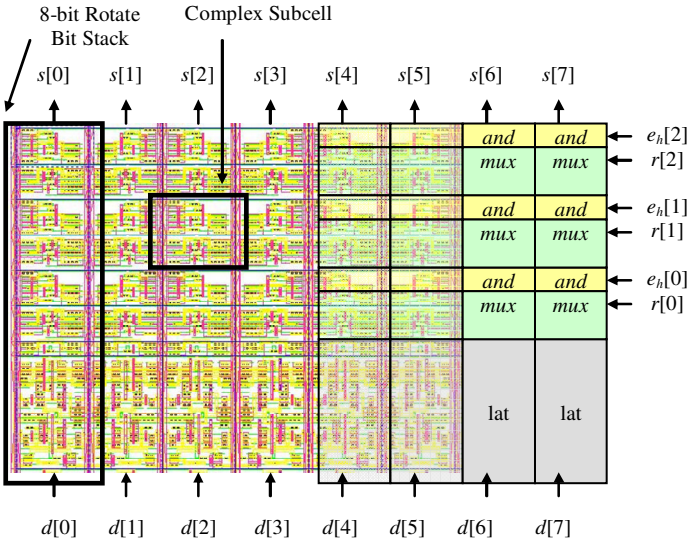


Figure 4. 8-Bit Rotate Physical Layout with 3-bit Encoding

Figure 5 displays the next level of hierarchy in which bit slices are placed next to each other to form a rotate row, n-bits wide. Let α denote the total latch height, β denote the total logic height of the stacked complex subcells (nine in this example, corresponding to r[0:8]), and ε denote the any added whitespace in the bit slice. Each bit stack is ordered, as in Figure 3, to line up the MUX rotate signals $r[i]$ and enable signals $e_h[i]$ and $e_v[j]$ with their corresponding complex subcell. This is critical for both routability and minimizing TWL, since the fanout on $r[i]$, $e_h[i]$ and $e_v[j]$ is very large. Between bit stacks (n/2 – 1) and (n/2), space for buffer placement is added where the rotate line bus $r[0{:}m\text{-}1]$ and

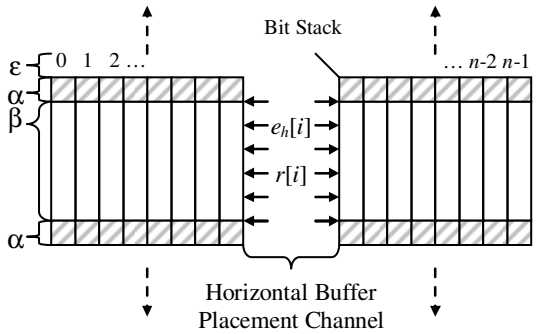


Figure 5. Rotate Row

enable signal bus $e_h[0{:}m\text{-}1]$ are lined up to drive horizontally to each bit slice.

The top level of hierarchy is shown in Figure 6 where each row from 0 to p-1 is an independent copy of the rotate row shown in Figure 5. In the middle of the block, space for buffer placement is added where the enable signal bus $e_v[j]$ is lined up to drive vertically to each row.

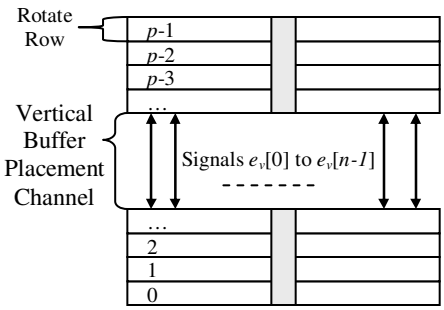


Figure 6. Fully Placed Rotate Block

3 DESIGN 2: AND/OR LOGIC

Design 2 is a standard AND/OR logic tree [8] [11] common throughout datapath design[1] with the bit stack logic structure shown in Figure 7. This structure is used in many applications, such as translation buffers and structured content addressable-memory circuits. Two signals, *a*_o and $s_d[j]$, are driven into an AND gate with the data inputs and then into an OR tree. The output of the OR tree is then ORed with a set signal e_i, and the result is latched.

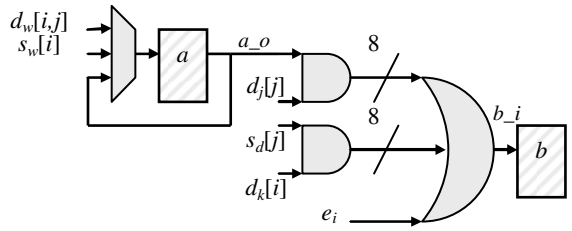


Figure 7. Design 2 Bit Stack Structure

3.1 Benchmark Details

Design 2 is a simplified version of a custom placed industrial design. Careful packing of the repeated logic enables optimization of both timing and area while reducing congestion. The bit stack in Figure 7 is repeated n=257 times in one row and there are m=32 rows placed within Design 2. Signal $d_w[i,j]$, $d_j[i]$, and $d_k[i]$, where i = 0, 1, …, n-1 columns and j = 0, 1, …, m rows, are primary data input signals; e_i and $s_d[i]$ are high fanout select lines running through the ith row where i = 0, 1, ..., n-1, and j = 0, 1, ..., m-1 for the bit stack with m rows and n columns. Select line $s_w[i]$ runs within row i and is a write enable select signal to latch new data into latch a. If $s_w[i]$ is not enabled, the prior value in a is selected and stored. Enable signal e_i is an override signal that will set latch b.

3.2 Placement Details

[1] Standard cell design practice allows for the interchange between logic gates of the same size. Thus, this implementation can be modified to many representative circuits, such as magnitude comparators, standard equality circuits or parity circuits.

Custom placement of Design 2 leads to regularly placed rows with tightly packed cells, shown in Figure 8, where eight total bit stack cells have been placed. Figure 8 represents a partial 8 bit stack lay out for illustrative purposes of the manual layout solution. In the full implementation, each bit stack consists of 16 AND gates driving a 16 way OR gate configuration. The logic gates from Figure 7 are interleaved into a single circuit row, and pins between rows for each select line are lined up evenly to reduce branch routing. Latch a and the MUX that drives it are placed at the bottom of the stack, the data flows through the AND/OR reduce logic.

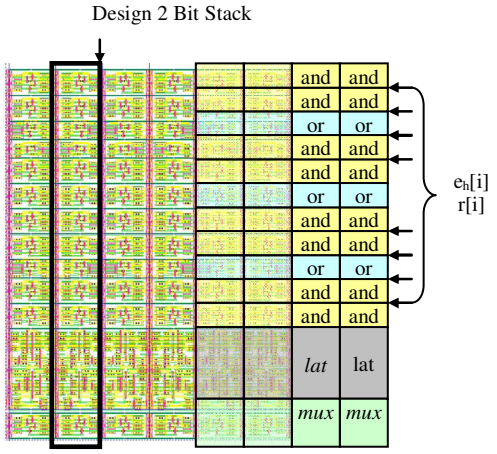


Figure 8. Design 2 Eight Bit Stack Physical Layout

Figure 9 shows the overall layout of the entire design with one bit stack shaded. Each bit stack is placed side-by-side n=257 times in one row, where there are 12 rows in total and placement space is added in the middle, both vertically and horizontally for the global wire drivers.

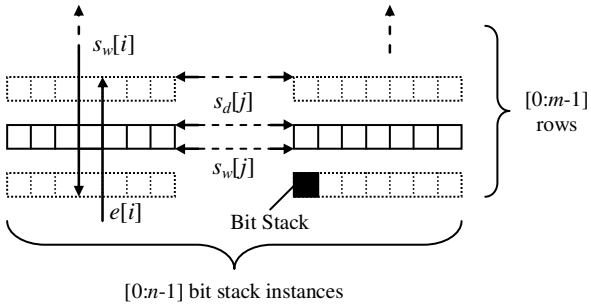


Figure 9. Placement for Design 2

4 EXPERIMENTAL RESULTS

Table 1 outlines the design details for each circuit that was constructed. Design 1 contains 140,800 movable cells, and Design 2 contains 130,944 movable cells. These are both reasonably small, custom-placement designs built using common structured placement tools, including schematic capture and layout. Once built, the netlist was exported to Bookshelf [14][15] format and the wirelength measured. The

custom layout solution is compared against the following placers for these two designs:

- mPL6 v6 [3]
- CAPO v10.2 [16]
- FastPlace v3.0 [20]
- NTUPlace3 v7.10.19 [4]
- APlace v1.0 [10]
- Dragon v3.01 [21]

Table 1. Design Characteristics

Designs	Num Nets	Num Pins	Num Terminals	# Movable Cells
DESIGN 1	157849	637984	19488	140800
DESIGN 2	148682	661340	21724	130944

The authors of Timberwolf [19] were contacted, but they were unable to provide a version compatible with the Bookshelf [14][15] format at this time. This simulated annealing approach may perform well on these moderately sized test cases. For all placers, a target density constraint[2] was not imposed to give maximum freedom to pack cells.

4.1 Initial Placement Results

Table 2 displays the results of the custom placement versus the academic placer. Column one shows all different placement algorithms, where "custom" corresponds to the manual-placed designs. Column two displays the measured TWL compared to the custom solution for each placement method on Design 1, and column five displays the TWL for Design 2. For both designs, APlace failed to find a legal placement solution. Columns three and six correspond to the percentage increases in TWL compared to the custom-placed solution. Columns four and seven display the placement runtime in seconds for each design.

Table 2. TWL Results

	Design 1			Design 2		
Placer	TWL	TWL Ratio	Run Time (s)	TWL	TWL Ratio	Run Time (s)
Custom	11000365	1.00	n/a	8642097	1.00	n/a
Capo	*15945589**	1.45	1453.9	*14381067**	1.66	1430.6
mPL6	*18290965**	1.66	n/a	n/a	n/a	n/a
ntuPlace3	n/a	n/a	n/a	10765222	1.25	533.0
APlace	n/a	n/a	n/a	n/a	n/a	n/a
Dragon	52926316*	4.81	2350.18	34711167	4.02	2692.0
FastPlace	*16336840**	1.49	194.9	n/a	n/a	n/a

*Completed with Overlaps
n/a entries did not complete for the base case

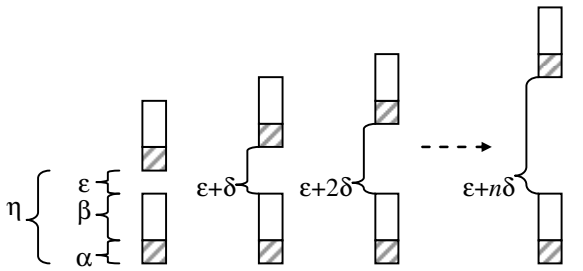
[2] This was achieved by supplying each placer with a target density requirement of 100% density as defined as in ISPD placement contests [14] [15]

The custom-placement method resulted in a TWL of 11,000,365 for Design 1 and of 8,642,097 for Design 2. For Design 1, all placers completed with overlaps. Of the placers that completed, CAPO produced the best automated placement result with 15945589, a 45% increase in TWL. The run time for all placers is less than fourty minutes because of the small design size. For Design 2, the ntuPlace3 algorithm resulted in the best automated placement result with 10,765,222, a 1.25 TWL ratio. The run time for all other placers is less than forty five minutes for Design 2.

4.2 Adding Additional Whitespace

As mentioned earlier, it is important to understand whether placers could not find the right structure, or could not legalize. Seven additional variations of each benchmark were generated by increasing white space using the following scheme. As shown in Figure 10, let η denote the total height of default bit slack, α denote the height of latch logic, β denote the height of placeable logic, and ε denote the white space added to the bit stack.

The original testcase experiment for both designs was set up with no extra white space between each row. Then the white space was increased using the scheme in Figure 10 and manually replaced the custom solution so that we could compare to the automated placement algorithms.



Total Cell Height: $\eta = \alpha + \beta + n\varepsilon$

Figure 10. Structured Placement Experiments

4.2.1 Whitespace Placement Results

Eight experiments were run on both designs, incrementally increasing the available whitespace to allow more room for automated placement tools while not applying any constraints.

Tables 3 and 4 display the TWL placement results for each experiment compared to the custom design solution at the same whitespace percentage. ntuPlace3 did not complete for Design 1 and APlace failed to legalize for both designs.

All placers except Dragon show wirelength improvement as more whitespace is added with a slight increase with the most whitespace. When more than 15% of whitespace is added however, the improvement of TWL and its ratio starts to

Table 3. TWL Results of Design 1

Placer	92.5	89.0	85.8	82.8	80.1	77.4	74.0	71.9
CAPO	1.45*	1.49*	1.24*	1.28*	1.14*	1.18*	1.12*	1.11*
mPL6	1.66*	1.65*	1.64*	1.66*	1.64*	1.66*	1.76*	1.73*
ntuPlace3	-	-	-	-	-	-	-	-
APlace*	-	-	-	-	-	-	-	-
Dragon	4.81	5.00	5.39	5.88	5.83	5.91	6.56	7.37
FastPlace	1.47*	1.33*	1.31*	1.31*	1.28*	1.26*	1.28*	1.31*

*Completed with Overlaps

saturate. After that point, adding additional white space did not significantly improve the overall TWL ratio compared to the custom placed design. Results for APlace are not shown because it failed to find a legal placement solution. One

Table 4. TWL Results for Design 2

Placer	96.1	93.6	89.5	85.3	81.5	78.1	75.2	72.2
CAPO	1.66*	1.24*	1.17*	1.18*	1.18*	1.20*	1.20*	1.21*
mPL6	-	1.19*	1.15*	1.72*	1.15*	1.16*	1.17*	1.18*
ntuPlace3	1.29	1.12	1.14	1.13	1.20	1.15	1.16	1.24
APlace*	-	-	-	-	-	-	-	-
Dragon	4.02	4.24	4.49	4.81	5.09	5.33	5.60	5.93
FastPlace	-	1.26	1.15	1.15	1.17	1.19	1.20	1.21

interesting result is the significant TWL increase in Dragon placer as available whitespace is added. In general, the overall *TWL ratios* are improving as whitespace increases however; this is primarily due to the TWL increase of the custom solution.

For Design 2, TWL at only 4% whitespace is significantly higher for all placers but quickly drops. The custom TWL for Design 2 increases significantly as whitespace increases for the design helping the overall TWL ratios for the placers. This again does not point to an improved placement solution with increased whitespace, but is instead a result of the logic spreading from the manual solution. Dragon placement results also exhibit the significant TWL increase trend seen in Design 1 as whitespace increases. Minimum overall TWL for the placers occurs within the range for 7% to 20% whitespace after which TWL begins to increase at a similar slope to the custom solution.

The following observations were made:

- Generally, placers did not improve much with additional white space, with the exception of CAPO, which improved from 1.66 to 1.17 on Design 2. Part of the improvement comes from the TWL increase from the manual solution.
- For Design 1, the best overall result for each experiment came from CAPO with a 14% increase in TWL at 80.1% utilization. The TWL increase of the manual solution is not obvious, but it also increases with added area.
- There was a gradual improvement in the TWL ratio for Design 2 as area increased for the design, with ntuPlace3 decreasing from 1.29 to 1.12.
- By industry standards, both designs are small relative to state-of-the-art work yet all placers presented significantly suboptimal TWL results.
- All placers failed to place Design 1 without overlaps. Though some of the placers completed Design 2 without overlaps, a 15% degradation in TWL translates to significant power and delay increases compared to a custom solution.

5 ANALYSIS

Obviously, it is disappointing that academic placers perform poorly on these real designs, so further examination is necessary. Figure 11 displays a zoomed in snapshot of three rows of the custom placed layout for Design 1 with the placeable logic between the latches from one row highlighted in light blue. The design was placed again using one of the better performing placement algorithms to see what happens to the blue cells that are densely packed in the custom layout. This is shown in Figure 12.

Observe the irregularity present in the blue highlighted logic. Most of the blue logic is placed within the bounds of the correct rows of latches, but a significant portion is left outside, despite adequate whitespace. This may occur because:

- Multilevel placement algorithms employ clustering to abstract a netlist into larger components that will be placed together. This manifests itself in loosely connected "blobs" of logic rather than densely packed structures.
- Analytical placement algorithms commonly employ net models such as cliques or stars of two-pin edges to represent hyperedges. Such models derate the weights of edges representing high-fanout nets to compensate for the increased number of edges needed to represent them. As a result, the impact of a 512-bit select line is very low, yet these nets could provide clues to the structure within the design.
- Clustering algorithms do not typically account for logic functions, and make decisions purely on local connectivity. This often leads to merging of gates across bit slices rather than merging the slice into a large cluster.

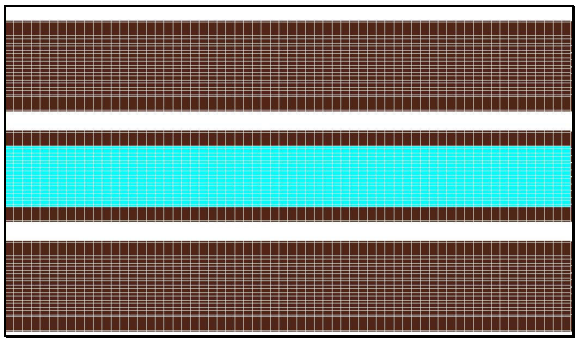


Figure 11. Custom Design 1 Placement Solution

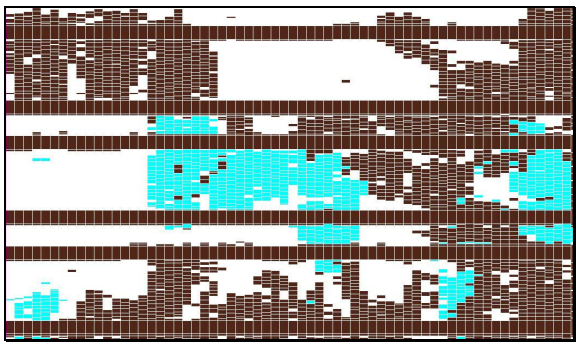


Figure 12. Automatic Design 1 Placement Solution

6 CONCLUSIONS

Recent years have seen truly significant improvements in runtimes, quality, and scalability in standard-cell placement algorithms. This work measures their performance on real datapath placement examples and compares them to hand-designed layouts. Academic placers still have a long way to go in order to match the quality of custom design solutions. An important contribution of this work is to release these benchmarks publically.

In order to keep pace with technology innovation, design automation must strive to improve productivity. This work highlights poor performance of modern placement tools on a key design style that is lacking in automation --- structured datapaths. One of the challenges posed by this problem is identifying regular datapaths within a larger design. Hence, new layouts will be constructed that combine both datapath and control logic in the same benchmark, providing even more challenging placement testcases. Future work will seek ways to improve automatic placement algorithms in terms of wirelength, routability, and timing closure of datapath placement problems.

7 REFERENCES

[1] S. N. Adya and I. L. Markov, *"Consistent Placement of Macro-blocks Using Floorplanning and Standard-Cell Placement'"* ISPD 2002, pp. 12-17.

[2] P.W. Bosshart and Q.D. An. *Shifter circuit for an arithmetic logic unit in a microprocessor.* US. patent 5896305, 1999.

[3] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. *mPL6: Enhanced multilevel mixed-size placement.* In Proc. ISPD, pp 212–214, 2006.

[4] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. *A high-quality mixed-size analytical placer considering preplaced blocks and density constraints,* In Proc. ICCAD, 2006.

[5] Chin-Chih Chang, Jason Cong, Michail Romesis, and Min Xie; *Optimality and Scalability Study of Existing Placement Algorithms;* IEEE Transactions on Computer-aided Design Vol..23, pp. 537-549 2004

[6] R.L. Davis. *Uniform shift networks.* Computer, 7:327-334, September 1974.

[7] Lars W. Hagen, Dennis J.-H. Huang, Andrew B. Kahng; *Quantified Suboptimality of VLSI Layout Heuristics Design Automation*, DAC pp 216-221, 1995.

[8] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[9] Hillebrand, M.A.; Schurger, T.; Seidel, P.-M.; *How to half wire lengths in the layout of cyclic shifters,* Fourteenth International Conference on VLSI Design, 2001, pp. 339 – 344.

[10] A. B. Kahng, S. Reda, and Q. Wang. *Architecture and details of a high quality, large-scale analytical placer.* In Proc. ICCAD, pp 890–897, 2005.

[11] Koren. *Computer Arithmetic Algorithms.* Englewood Cliffs, NJ: Prentice-Hall Inc., 1993.

[12] T. Machida. *Bidirectional barrel shift circuit.* US. Patent 4665538, 1987.

[13] S.M. Mueller and W.J. Paul. *Computer Architecture: Complexity and Correctness.* Springer Verlat, 2000.

[14] G.-J. Nam, *"ISPD 2006 Placement Contest: Benchmark Suite and Results,"* ISPD 2006, p. 167.

[15] G.-J. Nam, C. J. Alpert, P. G. Villarrubia, B. B. Winter, M. C. Yildiz *"The ISPD2005 Placement Contest and Benchmark Suite,"* ISPD 2005, pp. 216-220.

[16] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. *Min-cut floorplacement.* IEEE Transactions on Computer-Aided Design, Vol. 25, pp. 1313–1326, July 2006.

[17] P.-M. Seidel. *On the Design of IEEIT Compliant Floating-point Units and Their Quantitative Analysis.* PhD thesis, University of Saarland, December 1999.

[18] L. Sigal *et al. Circuit design techniques for the high-performance CMOS IBM S/390 Parallel Enterprise Server G4 microprocessor.* IBM Journal of Research and Development, Vol. 41, pp. 489-503, 1997.

[19] Wern-Jieh Sun and Carl Sechen, *Efficient and Effective Placement for Very Large Circuits*, IEEE Transactions on Computer-Aided Design, Vol 14, pp. 349-359, 1995

[20] Natarajan Viswanathan and Chris Chong-Nuen Chu, *FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement, and a Hybrid Net Model* IEEE Transactions on Computer-Aided Design, VOL. 24, pp. 722-733 2005

[21] M. Wang, X. Yang, and M. Sarrafzadeh, *Dragon2000: Standard-cell placement tool for large industry circuits,* in Proc. IEEE/ACM Int. Conf. Computer-Aided Design, 2000, pp. 260–263.

ISPD'11, March 27-30, 2011, Santa Barbara, California, USA.