IMPLICATIONS OF AREA-ARRAY I/O FOR ROW-BASED PLACEMENT METHODOLOGY*

A. E. Caldwell, A. B. Kahng, S. Mantik and I. L. Markov
UCLA Computer Science Dept., Los Angeles, CA 90095-1596
{caldwell,abk,stefanus,imarkov}@cs.ucla.edu

ABSTRACT

We empirically study the implications of area-array I/O for placement *methodology*. Our work develops a three-axis testbed that examines (1) I/O regime (area-array vs. peripheral pad locations), (2) I/O and core placement methodology (variants of alternating vs. simultaneous I/O and core placement approaches), and (3) placement engine (hierarchical quadratic for both core and I/O cells vs. pure min-cut for core cells and assignment for I/O). Experimental data show that the area-array I/O regime is somewhat "more forgiving" of bad placement methodologies than the peripheral I/O regime. On the other hand, the wrong methodology can still entail substantial losses in solution quality and efficiency. Last, we hypothesize that reductions of on-chip wirelength from adopting the area-array I/O regime may be correlated with topological depth of circuits.

1. INTRODUCTION

IC packaging technologies with peripheral I/O pads have well-known shortcomings. Observed system Rent parameters suggest that ICs require asymptotically more pads than the die perimeter can provide [15]. Peripheral I/O pads also constrain clock/power distribution, and their inherently large parasitics cause coupling and power issues for off-chip signaling. Given these concerns, the area-array I/O regime is projected to eventually dominate IC implementation methodology, affording improved pad count and reliability, reduced noise coupling, and cost savings as the technology matures.

With respect to physical design, area-array I/O presents several critical new issues. Locating pads directly over the core layout region requires new tools for signal integrity analysis and simultaneous die-package simulation. The possibility of synergetic on-chip and on-package routing opens up many new problems, particularly in the distribution of clock and power. Finally, place-and-route tools will need to handle new layout constraints and objectives, e.g., for noise decoupling and ESD protection. Much effort remains in developing tools and methodology, and the precise impact of area-array I/O on such basic parameters as routability, required global interconnect resource for the design, etc. remains largely unknown.

In this work, we focus on the implications of the area-array I/O regime on I/O and core placement *methodology* for row-based ICs. Specifically, we examine whether the shift to area-array I/O requires changes to methodologies (e.g., alternating I/O and core placement) and engines (e.g., top-down "quadratic placers" that integrate sparse-system solvers with min-cut partitioners) that are perceived to be successful in the peripheral I/O regime. Our specific contributions include:

- We develop a three-axis testbed that allows examination of (1) I/O regime (area-array vs. peripheral pad locations), (2) I/O and core placement methodology (variants of alternating vs. simultaneous I/O and core placement approaches), and (3) placement engine (hierarchical quadratic for both core and I/O cells vs. pure min-cut for core cells and assignment for I/O).
- Under modest assumptions, we experimentally show that area-array I/O leads to smaller wirelengths (suggesting better routability) than peripheral I/O.
- Again experimentally, we show that the area-array I/O regime is somewhat "more forgiving" of bad I/O and core placement methodologies. On the other hand, the wrong methodology can still hurt solution quality and/or runtime.

The remainder of this paper is organized as follows. In Section 2, we review related previous work, while Section 3 gives a model of I/O regimes, along with a taxonomy of I/O and core placement methodologies and placement engines. Section 4 describes details of our experimental testbed, and Section 5 gives experimental studies of different placement methodologies in the peripheral and area-array I/O regimes. Section 6 concludes with a description of ongoing research efforts.

2. PREVIOUS WORK

For the intrinsic area-array regime,[1] to which our studies apply, Tan et al. [17] pose the layout problem as: "Given a core portion of the chip which already contains the I/O buffers, place the possible uniformly spaced area-array pads on the top metal layer of the design ... and route these pads to the I/O ports of the chip...". For a given fixed core placement, Tan et al. propose a pad placement and assignment methodology that searches for feasible pad locations until there are 10% more identified pad locations than I/O ports. Assignment is used to match I/O ports to pad locations. Farbarik et al. [5] report a suite of CAD tools for intrinsic area-array ICs, notably an area-pad floorplanner that drives the placement of blocks and associated

*THIS WORK WAS SUPPORTED BY A GRANT FROM CADENCE DESIGN SYSTEMS, INC.

[1] *Intrinsic* area-array ICs are those originally designed and laid out for area-array bonding; *extrinsic* area-array ICs are originally designed and laid out for peripheral bonding, and later converted to the area-array regime using a signal redistribution layer [17]. Maheshwari et al. [11] distinguish the same concepts as "true area-I/O" vs. "redistribution".

I/O buffers so as to minimize routing cost to area pads.[2] A corner-stitched maze router is used to perform the area-pad routing. Kiamilev et al. [9] propose three distinct layout methodologies for intrinsic area-array ICs. The second approach is similar to that of [5] but uses manual routing; the third approach separately optimizes the core and I/O buffer circuit placements on chip, then places area pads directly above corresponding I/O buffers using a two-dimensional array structure called a *pad interface module*.

An important work vis-a-vis our own is that of Maheshwari et al. [11], which addresses timing and wirelength implications of transitioning a field-programmable MCM implementation to the area-array I/O regime. In [11], analysis of a theoretical model based on random placement and Rent's rule suggests that area-array I/O "offer[s] a small improvement on the total wirelength, but not much", since the proportion of external nets decreases as the circuit grows large. On the other hand, experimental results with an FPGA layout tool indicate that area-array I/O can afford increased routability (around 13% average reduction in wiring resource usage) and smaller parasitics, particularly for combinational benchmarks.

Finally, I/O placement methodology has been extensively studied for row-based designs and the peripheral I/O regime. RITUAL [16] leaves I/Os floating on the core boundary (allowing infeasible positions) and legalizes them during detailed placement. PACT [13] notes the potential weak I/O placement quality of RITUAL, and proposes linear assignment to compute an initial (i.e., before core cells are placed) I/O placement. Analysis of circuit structure and path timing constraints is used to determine assignment costs. The authors of PACT [13] observe, by way of motivation, that the common practice of alternating I/O and core placements starting from an arbitrary initial I/O placement can be time-consuming; furthermore, "even if convergence is achieved, the final solution is heavily influenced by the [arbitrary] initial pad assignment". For small instances of up to 589 gates and 301 I/Os, PACT achieves between 6.5% and 23.6% percent reduction in total wirelength versus random I/O placement, and between 1.0% and 9.7% improvement in path timing.

Chen and Marek-Sadowska [2] note possible weaknesses in PACT, e.g., that the chip boundary is represented by a linear array of locations. The authors of [2] construct an initial I/O placement by annealing, using circuit structure and path delay constraints in the objective. The path timing penalty of a bad I/O placement is estimated as being up to 10%; no wirelength penalty data is given. A significant improvement to PACT is given by [14], who separately process *floating inputs* in performing the initial I/O placement. When both methods are compared in the context of a wirelength-driven annealing placer, the method of [14] improves over PACT total wirelength by 42% for two of the four cases reported (with up to 412 cells and 59 pads). The work of Gao et al. [7] typifies top-down partitioning-based placers that deal with peripheral I/O pads and core cells *simultaneously*, i.e., separate balance factors are maintained for I/Os and core cells during the partitioning step.

[2] The netlist assigning I/O buffers to pads is assumed fixed, in contrast to [17] where "[the] netlist is generated using a pad assignment method...".

3. A TAXONOMY OF I/O AND CORE PLACEMENT METHODOLOGY

3.1. I/O Regimes

We study the following abstract model of I/O regimes for row-based designs.

- I/O cells must be placed exactly at pad locations, and any I/O cell can be placed at any pad location.
- No two I/O cells can occupy the same location.
- For a design with P I/O cells and a rectangular core layout region, we fix pad locations:
  - with P locations uniformly spaced around the boundary of the core layout region (this models the peripheral I/O regime with *generic locations*),
  - with P locations along the sides of the core layout region determined by "projecting" original peripheral pad locations[3] to the layout boundary with respect to the center of the layout region (this models the peripheral I/O regime with *user locations*), or
  - with an array of locations spaced uniformly within the core layout region (this models the area-array I/O regime).[4]

Our model of the area-array regime ignores many practical constraints, as well as freedoms, that are associated with pad location and I/O assignment to pads. For example, we ignore I/O placement constraints – many of which have yet to be precisely characterized – that arise from package technology, decoupling and ESD requirements, power/ground distribution, etc. However, our model permits controlled study of netlist embeddability and embedding strategy, and is tractable even to industry layout tools that do not otherwise handle the area-array regime. Within this model, we investigate the following taxonomy of methodologies and engines for I/O and core placement.

3.2. Alternating vs. Simultaneous Methodologies

Previous works use two basic methodologies, *alternating* and *simultaneous*, for I/O and core cell placement.

- **Alternating I/O and core placement.** The alternating methodology iterates between two basic steps, (1) fixing core cells and re-placing I/Os, and (2) fixing I/Os and re-placing core cells. There are two distinct variants, *alternating core-first* and *alternating I/O-first*. In the core-first variant, we delete all I/O cells from the netlist, then place the core cells into legal core sites to minimize, e.g., a total wirelength objective. We then restore the original netlist, and begin the alternation with step (1). In the I/O-first variant, we determine an initial I/O placement and then begin the alternation with step (2). As noted above, I/O-first alternation typically uses a *random* initial I/O placement. In the peripheral I/O regime, we can also have a *user-defined* initial I/O placement.
- **Simultaneous I/O and core placement.** The simultaneous methodology determines placements of I/O and core cells at the same time. For example, [7] performs hierarchical top-down placement, and specifies that any physical partition at any level contains (assignable) I/Os in proportion to the number of (available) legal pad sites contained in the partition.

[3] I.e., as specified in the benchmark data when we received it.

[4] If the core region has dimensions $H \times W$, there will be $\frac{H}{2(H+W)}$ rows and $\frac{W}{2(H+W)}$ columns of pads.

3.3. Hierarchical Placement Engines

Most placement tools, with the notable exception of TimberWolf [18], are reputed to incorporate the hierarchical top-down strategy as part of their approach. The two most common variants are based respectively on *analytic placement* and *pure min-cut placement*.

- **Analytic placement.** As reviewed in [1], the *analytic placement* (popularly referred to as "quadratic placement") approach involves (1) solution of a sparse linear system to determine a continuous module placement that minimizes a squared-wirelength objective[5], interleaved with (2) heuristic means of "legalizing", or "spreading", the placement so that it satisfies discrete constraints (e.g., no module overlaps). The sparse linear systems can correspond to hierarchical subproblems in the top-down placement process. The legalization step is typically accomplished via partitioning or assignment/transportation formulations, along with highly sophisticated metaheuristics.
- **Pure min-cut placement.** A pure min-cut strategy uses an iterative partitioner, e.g., some variant of the Fiduccia-Mattheyses heuristic [6], with a minimum net cut objective. Techniques such as terminal propagation, table-lookup treelength estimators, etc. are used to improve fidelity of the min-cut partitioning objective with respect to the actual placement objective, which is often based on the sum of net bounding box half-perimeters. Again, many sophisticated metaheuristics are used to achieve competitive results.

4. EXPERIMENTAL QUESTIONS AND TESTBED

In most IC place-and-route flows today, I/O locations are assumed to be fixed in any input instance to the placement tool. For example, in hierarchical blocks the I/O locations may be fixed by a pin optimizer or by a chip-level route planner within the design planning tool. Even for "flat, full-chip" placement (e.g., traditional gate-array ASIC), CAD engineers often heuristically assign system I/Os to pad locations. The conventional wisdom is that fixing the I/O placement before placing core cells impacts the total wirelength of the layout by at most a few percent. This is plausible since the number of nets incident to I/Os is small relative to the total number of nets, and matches the conclusions of [11]. On the other hand, recall that in the peripheral I/O regime (i) a high-quality I/O placement can significantly improve wirelength and performance for small designs [2], and (ii) with alternating I/O and core placement methodologies the final solution is "heavily influenced" by the initial I/O placement [13]. Thus:

- **Experimental Question 1:** Does an initial I/O placement have as "heavy" an influence (with respect to alternating methodologies) in the area-array I/O regime as it does in the peripheral I/O regime?

Next, we observe that within the hierarchical placement approach, the use of analytic placers ("quadratic placers") *requires* fixed I/O locations to "anchor" the result of the continuous placement. If there are no prescribed I/O locations, then the analytic placement collapses down to a single point and provides a vacuous start to the legalization step. We ask whether a "fix I/Os first" mindset with the analytic placement approach has somehow guided I/O and core placement methodology in the wrong direction.

- **Experimental Question 2:** Can core-first alternation (which a pure min-cut placement engine might achieve more naturally than a quadratic placer) deliver results superior (in terms of solution quality and/or convergence) to those of I/O-first alternation?
- **Experimental Question 3:** Can a simultaneous methodology deliver results superior (in terms of solution quality and/or convergence) to those of an alternating methodology?

Each of these questions can be asked in both the peripheral and area-array I/O regimes.

Testbed

To test all placement strategies within the above taxonomy, we require four capabilities: (1) placement of core cells with I/Os detached from the netlist; (2) placement of core cells with fixed I/Os; (3) placement of I/Os onto discrete locations when core cells are fixed; and (4) simultaneous placement of I/O and core cells.

Our testbed includes both an industry placer and our own placer. The industry placer reputedly uses some form of the quadratic (analytic) placement approach; we run it in default mode. It can be coerced to provide the first three out of the four capabilities above, via a mix of site definitions, fixed-location constraints, I/O-core reclassification, and scripting. Our placer integrates a number of techniques, but we invoke only pure hierarchical min-cut placement for core cells and min-cost assignment for I/Os to achieve all four of the capabilities above.

Both the industry placer and our placer can read test cases from industry in the Cadence LEF/DEF format. Our experiments are based on two industry designs from which we removed all nets incident to more than 100 cells, as well as all spare cells and the 1-pin nets that result from removal of spare cells. Parameters of the test cases are shown in Table 1.

Test Case	Cells	I/Os	Nets
Case 1	3231	545	2844
Case 2	12133	662	11826

Table 1. **Test cases used for experiments**.

5. EXPERIMENTAL RESULTS

5.1. Experiments With the Industry Placer

The industry placer, to our knowledge, assumes that I/Os are placed and fixed. We have not yet been able to achieve a simultaneous methodology with this tool. To implement alternating I/O and core placement methodologies, we modify the netlist at each iteration by relabeling former movable core cells as fixed I/Os, and former fixed I/Os as movable core cells. The placer can also be run with all I/Os removed from the netlist to construct initial core placements. Notice that the placement of core cells with all I/Os removed from the netlist gives an empirical "lower bound" for wirelength. Such lower bounds are 246633 microns for Case 1 and 1602701 microns for Case 2.

Table 2 presents placement results (sum of net bounding box half-perimeters) for the I/O regimes discussed in the

[5] In some sense, the term "quadratic placement" is a misnomer since it is also possible to analytically solve for a continuous placement that minimizes a different objective, e.g., linear wirelength.

Industry Placer						
Case 1 (WL $\times 10^5 \mu m$)						
I/O Regime	Alternation	1^{st} Iter all	1^{st} Iter I/O	Best Iter all	Best Iter I/O	#
PG	Core	4.75	2.60	4.17	1.55	4
	I/O(r)	6.62	3.42	4.43	1.65	19
PU	Core	5.41	3.26	4.68	1.98	6
	I/O(r)	6.79	3.44	4.92	2.10	14
	I/O(u)	6.44	3.07	4.83	2.06	17
AA	Core	2.84	0.71	2.82	0.68	6
	I/O(r)	5.77	2.59	3.11	0.72	19
Case 2 (WL $\times 10^6 \mu m$)						
PG	Core	2.15	0.59	2.04	0.35	14
	I/O(r)	2.71	0.85	2.05	0.33	16
PU	Core	2.31	0.73	2.16	0.44	6
	I/O(r)	2.78	0.88	2.16	0.44	16
	I/O(u)	2.81	0.88	2.14	0.38	7
AA	Core	1.70	0.14	1.67	0.12	16
	I/O(r)	2.54	0.66	1.76	0.13	19

Table 2. **Studies of peripheral and area-array I/O regimes.**

previous section: peripheral with generic locations **(PG)**, peripheral with user-defined locations **(PU)**, and area-array **(AA)**. Methodologies studied were alternating starting with a fixed initial core placement **(Core)** and alternating starting with a fixed initial I/O placement **(I/O)**. Recall that the PU,I/O combination permits use of the designer's I/O placement. Thus, for the PU regime we distinguish a starting *random* I/O placement **(I/O(r))** from a starting *user-defined* I/O placement **(I/O(u))**. All experiments involving random initial I/O placements (I/O(r)) are averages over 5 runs.

We place I/Os and core 10 times each, in alternation, using the industry placer; this yields 19 iterations with meaningful placement results.[6] We report best wirelength values over all 19 iterations along with the iteration number (1-19) at which the best result occurred. We also report wirelength values obtained at the first iteration. Two wirelength values are given: summed over all nets, and summed over only nets that contain I/Os.

For the standard methodology of alternating I/O and core placement steps starting with a random I/O placement, surprisingly many iterations are required until the best placement is reached. (Quite possibly, current practice does not invoke sufficiently many alternations.) Often, wirelength does not improve monotonically over successive iterations, but instead oscillates. This is shown in Figure 1, which shows all results in the PG and AA regimes for both Case 1 and Case 2. The oscillating effect is very strong with peripheral I/O regimes, and relatively weak with area-array I/O. Furthermore, the I/O placement steps for peripheral I/O usually increase wirelength, while core placement steps usually decrease wirelength. One possible explanation is that the industry placer's algorithm, while highly successful for core placement, does not perform as well for I/Os. Thus, the I/O placement iterations are essentially "perturbations" or "kick moves" from which the core placement recovers. We also note that the best wirelengths in all I/O(r) experiments obtain from 22% to 46% reduction over the initial wirelengths. We conclude that an initial random I/O placement can be very harmful, and that the traditional methodology may require many alternations to recover.

[6] On a 140MHz Sun Ultra-1, average CPU time per core placement iteration was 2 minutes and 20 minutes, and average CPU time per I/O placement iteration was 45 seconds and 2.5 minutes, for Cases 1 and 2 respectively.

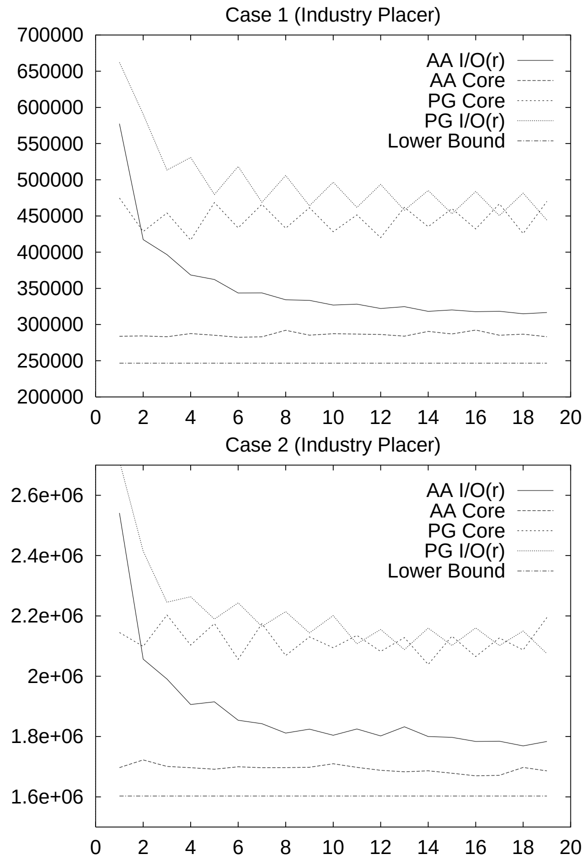

Figure 1. **Wirelengths after I/O and core placement iterations** for Case 1 (top) and Case 2 (bottom) with the industry placer.

We next observe that with every placement regime, placing core cells first with I/Os removed leads to overall smaller wirelength. Furthermore, in all but the PU,I/O(u) case, far fewer iterations are required before essentially the best wirelength is achieved. From a methodology point of view, it appears that disconnecting the I/Os from the netlist and placing the core yields a superior start for alternation. Finally, total wirelength values show that the best I/O regime is area-array, followed by uniformly-spaced (generic) peripheral. The worst regime in terms of "achievable" wirelength was the PU regime: we speculate that the nonuniformity of the pad locations makes wirelength minimization more difficult for the various methodologies. Notice that in the PU regime, the I/O placement defined by the designer leads to overall smaller wirelength, again confirming that the initial I/O placement strongly influences the final placement and that a bad initial placement is harmful.

5.2. Experiments With Our Placer

With our placer, a simultaneous methodology is implemented by performing top-down hierarchical placement, with core cells recursively bipartitioned by a min-cut KLFM heuristic at each level, and I/O cells re-placed by min-cost assignment at each level. To set the assignment costs for the I/O cells, we assume that each core cell is located in the center of its current region within the top-down process.[7] Our implementation of alternating methodologies uses top-down min-cut FM bipartitioning for each core placement iteration, and min-cost assignment for each I/O placement iteration. With our placer, all reported results are averages over 5 runs.

Our Placer							
Case 1 (WL $\times 10^5 \mu m$)							
I/O Regime	Methodology	1^{st} Iter all	I/O	Best Iter all	I/O	#
PG	Core	3.63	1.66	3.25	1.17	17
	I/O	6.41	3.88	3.45	1.28	18
	Simultaneous	—	—	4.05	1.69	—
AA	Core	2.56	0.62	2.56	0.62	1
	I/O	5.52	3.12	2.68	0.63	12
	Simultaneous	—	—	3.00	0.82	—
Case 2 (WL $\times 10^6 \mu m$)						
PG	Core	1.79	0.25	1.73	0.16	15
	I/O	2.64	0.94	1.85	0.22	16
	Simultaneous	—	—	1.89	0.25	—
AA	Core	1.68	0.13	1.66	0.11	7
	I/O	2.41	0.76	1.71	0.12	14
	Simultaneous	—	—	1.74	0.13	—

Table 3. **Studies of peripheral and area-array I/O regimes.**

Table 3 shows that as the total wirelength improves from the first iteration to the best iteration, the same reduction (or even greater reduction) occurs in the nets incident to I/Os. This phenomenon appears to be independent of I/O regime and placement methodology. As expected, using min-cost assignment for I/O placement results in much more efficient I/O placement iterations, and generally decreases wirelength from the previous iteration (in contrast to what we observed with the industry placer).[8] As seen in Figure 2, wirelength decreases faster than with the industry placer. Furthermore, near-best wirelength results tend to be achieved in fewer iterations (sometimes almost immediately). We believe that this is due to use of assignment for I/O placement.

In general, alternating methodology experiments with our placer lead to similar qualitative conclusions as experiments with the industry placer. We emphasize that our wirelength values and CPU times are not comparable with those of the industry placer, since the industry placer performs many detailed optimizations to guarantee routability. (We leave such optimizations, and the ability to perform direct comparisons between the two placers, to future work.) Finally, we note that the results from the simultaneous methodology are fairly disappointing. We believe that variant implementations may be more successful, and our future work targets these.

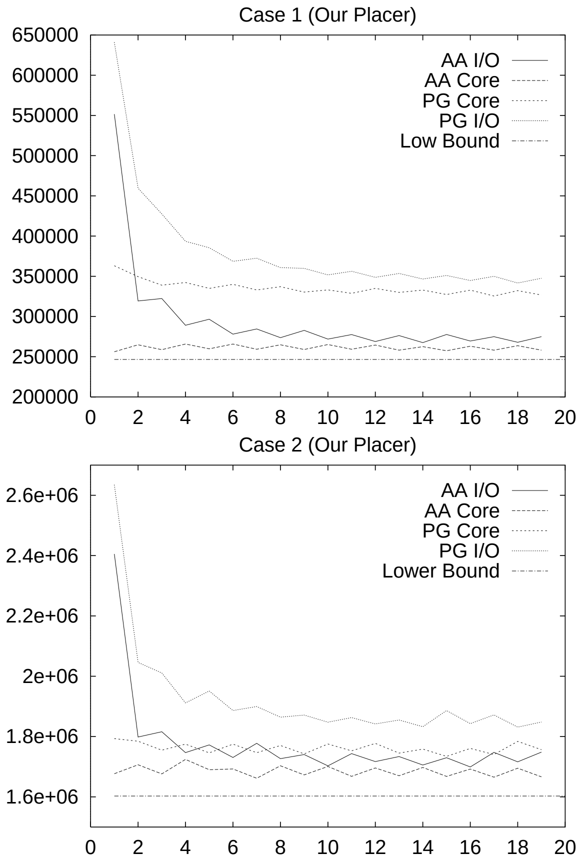

Figure 2. **Wirelengths after I/O and core placement iterations** for Case 1 (top) and Case 2 (bottom) with our placer.

5.3. Experiments with Netlist Structure

Our final experiments seek a link between topological structure of netlists and the ability of area-array I/O to reduce total wirelength. We conjecture that designs in which core cells are nearer I/Os will benefit greatly from area-array I/O, while designs in which core cells are further from I/Os will not benefit as much. The intuition is that area-array I/O offers more flexibility in the I/O placement, and that this can help a design whose core cells are tightly bound to I/Os. On the other hand, if the core cells are loosely bound to I/Os, then extra flexibility in the I/O placement may not be needed to achieve low wirelength.

To verify this conjecture, we have produced a series of *mutant* netlists by (1) iteratively finding the core cells furthest from I/Os (using simple hop-count in the netlist hypergraph), and (2) transforming these cells into "I/Os". In this way, we may change the topological depth fairly quickly with minimal change to the netlist. Table 4 shows the *topological profile*, i.e., the number of cells at each hop-count (level) from the I/Os, for the each original design and five mutants selected for their variety of topological depths. Each mutant is designated by the number of cells changed from the original (e.g., Case 1m40 has 40 original core cells changed into I/Os). For each mutant, we perform the same placement experiments as before, but apply only the most successful methodology, i.e., alternating core-first. These

[7]Nets containing multiple I/Os are not common in our experience, hence I/O placement can be reduced to the classical assignment problem with independent assignment costs for each combination of I/O and pad location. Our implementation uses the efficient cost scaling algorithm due to Goldberg and Kennedy [8].

[8]On a 140MHz Sun Ultra-1, average CPU time per core placement iteration was 2 minutes and 15 minutes, and average CPU time per I/O placement iteration was 14 seconds and 22 seconds, for Cases 1 and 2 respectively.

Test Case	I/Os	Topological profile (by levels)
Case 1	545	1030, 1265, 352, 33, 6
Case 1m6	551	1037, 1272, 345, 26
Case 1m19	564	1072, 1358, 237
Case 1m40	585	1253, 1300, 93
Case 1m100	645	1433, 1143, 10
Case 1m110	655	1459, 1117
Case 2	662	1010, 2668, 4194, 2667, 786, 137, 9
Case 2m3	665	1019, 2685, 4244, 2727, 706, 87
Case 2m10	672	1054, 2835, 4522, 2633, 410, 7
Case 2m16	678	1069, 2877, 4599, 2570, 340
Case 2m40	702	1170, 3353, 5068, 1802, 38
Case 2m67	729	1247, 3604, 5083, 1470

Table 4. **Topological profiles** of original and mutant netlists for Case 1 and Case 2. The topological profile indicates the number of cells at each hop-count (level) from the I/Os.

results are shown in Table 5.

To assess the impact of the area-array regime, we observe the difference between best wirelengths achieved with area-array I/O and with peripheral I/O. For the smaller design, Case 1, this difference clearly increases as more cells in the design move closer to the I/Os. However, for Case 2 only minor changes in this difference can be seen. We believe that alternate characterizations of netlist topology may be needed, as well as more detailed studies.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have empirically studied the implications of area-array I/O for placement *methodology* using a testbed that allows us to vary the I/O regime, the I/O and core placement methodology, and the placement engine. Our main results are as follows.

- We confirm that with alternating placement methodologies, which are often used in practice today, the number of iterations needed to achieve good solutions can be surprisingly large. Also, a bad (e.g., random) initial I/O placement can seriously handicap subsequent iterations.
- We also conclude that it is better to begin the alternation by placing a netlist of core cells with all I/Os disconnected, rather than by placing I/Os randomly as in traditional approaches. This has interesting implications vis-a-vis the use of quadratic placers, which can require fixed "anchors" in the placement instance.
- Our experiments also show that the area-array I/O regime offers substantial wirelength improvements over the best methodology we found for the peripheral I/O regime. The wirelength reductions range from 12%-27% with an industry placer, and average around 30% with our placer.
- We show that the classical assignment problem may be a more appropriate framework for the I/O placement iteration. We observe that the assignment formulation accurately captures placement costs since I/Os rarely share nets, and that assignment algorithms seem more efficient than traditional placers on I/Os-only instances. Furthermore, traditional placers may not easily adapt to I/O placement instances, and may even increase the total wirelength in some iterations.

Industry Placer					
Case 1 (WL $\times 10^5$)					
Mutant Case	Pad Regime	1^{st} Iter all	1^{st} Iter I/O	Best Iter all	Best Iter I/O
1	PG	4.75	2.60	4.17	1.55
	AA	2.84	0.71	2.82	0.68
1m6	PG	4.62	2.46	4.17	1.53
	AA	2.88	0.72	2.80	0.65
1m19	PG	4.63	2.57	4.09	1.68
	AA	2.90	0.84	2.78	0.69
1m40	PG	4.81	2.90	4.17	1.68
	AA	2.67	0.75	2.62	0.67
1m100	PG	5.06	3.24	4.53	2.23
	AA	2.74	0.92	2.64	0.81
1m110	PG	5.14	3.34	4.61	2.35
	AA	2.79	0.99	2.70	0.85
Case 2 (WL $\times 10^6$)					
2	PG	2.15	0.59	2.04	0.35
	AA	1.70	0.14	1.67	0.12
2m3	PG	2.18	0.60	2.02	0.34
	AA	1.73	0.15	1.73	0.13
2m10	PG	2.08	0.50	1.95	0.31
	AA	1.74	0.16	1.74	0.16
2m16	PG	2.14	0.57	1.94	0.32
	AA	1.74	0.16	1.68	0.13
2m40	PG	2.13	0.57	1.98	0.33
	AA	1.71	0.15	1.66	0.13
2m67	PG	2.21	0.59	2.04	0.38
	AA	1.79	0.17	1.67	0.13
Our Placer					
Case 1 (WL $\times 10^5$)					
Mutant Case	Pad Regime	1^{st} Iter all	1^{st} Iter I/O	Best Iter all	Best Iter I/O
1	PG	3.63	1.66	3.25	1.17
	AA	2.56	0.62	2.56	0.62
1m6	PG	3.66	1.69	3.25	1.17
	AA	2.64	0.66	2.63	0.63
1m19	PG	3.67	1.73	3.28	1.23
	AA	2.69	0.76	2.64	0.66
1m40	PG	3.67	1.82	3.30	1.38
	AA	2.64	0.79	2.60	0.70
1m100	PG	3.92	2.17	3.49	1.65
	AA	2.70	0.94	2.57	0.79
1m110	PG	4.02	2.25	3.54	1.73
	AA	2.77	0.99	2.61	0.82
Case 2 (WL $\times 10^6$)					
2	PG	1.79	0.25	1.73	0.16
	AA	1.68	0.13	1.66	0.11
2m3	PG	1.78	0.24	1.74	0.17
	AA	1.67	0.13	1.68	0.11
2m10	PG	1.79	0.26	1.75	0.18
	AA	1.67	0.13	1.66	0.12
2m16	PG	1.80	0.26	1.74	0.17
	AA	1.74	0.16	1.66	0.12
2m40	PG	1.80	0.28	1.76	0.20
	AA	1.68	0.15	1.68	0.15
2m67	PG	1.82	0.31	1.76	0.21
	AA	1.68	0.16	1.67	0.14

Table 5. **Studies of peripheral and area-array I/O placement regimes with netlist mutants.**

Our ongoing work addresses three issues. First, our experiments have shown that our combination of min-cut and assignment may eventually be competitive with the industry placer, at least in situations where both I/O and core cells must be placed. (Again, recall that our approach does not require any fixed anchors, and that the first iteration appears very important.) We hope to improve our placer to meet similar routability and legalization objectives as the industry tool, so that more direct comparisons will be possible. Second, we believe that the simultaneous I/O and core placement methodology still holds promise, and we hope to find a better implementation. Finally, we hope to better understand the relationship between netlist topology and wirelength reductions afforded by the area-array regime.

REFERENCES

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

[2] B. Chen and M. Marek-Sadowska, "Timing Driven Placement of Pads and Latches", *Proc. IEEE International ASIC Conf.*, Rochester, Sept. 1992, pp. 30-33.

[3] B. T. Davis, C. Gauthier, P. Parakh, T. Basso, C. Lefurgy, R. Brown and T. Mudge, "Impact of MCM's on High Performance Processors", *Proc. INTERpack-97: International, Intersociety Electronic and Photonic Packaging Conf.*, Mauna Lani, vol. 1, June 1997, pp. 863-868.

[4] P. Dehkordi, C. Tan and D. Bouldin, "Intrinsic Area Array ICs: What, Why and How", *Proc. IEEE Multi-Chip Module Conf.*, Santa Cruz, Feb. 1997, pp. 120-124.

[5] R. Farbarik, X. Liu, M. Rossman, P. Parakh, T. Basso and R. Brown, "CAD Tools for Area-Distributed I/O Pad Packaging", *Proc. IEEE Multi-Chip Module Conf.*, Santa Cruz, Feb. 1997, pp. 125-129.

[6] C.M Fiduccia and R.M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.

[7] T. Gao, C. L. Liu and K. C. Chen, "A Performance Driven Hierarchical Partitioning Placement Algorithm", *Proc. European Design Automation Conf.*, Hamburg, Sept. 1993, pp. 33-38.

[8] A. Goldberg and R. Kennedy, "An Efficient Cost Scaling Algorithm for the Assignment Problem", *Math. Prog.* 71 (1995), pp. 153-178.

[9] F. E. Kiamilev, A. V. Krishnamoorty, J. Rieve, R. G. Rozier, G. F. Alpin, C. D. Hull, R. Farbarik and R. E. Oettel, "Design of ICs for Flip-Chip Integration with Optoelectronic Device Arrays", *Proc. IEEE Multi-Chip Module Conf.*, Santa Cruz, Feb. 1997, pp. 163-167.

[10] R. J. Lomax, R. B. Brown, M. Nanua and T. D. Strong, "Area I/O Flip-Chip Packaging to Minimize Interconnect", *Proc. IEEE Multi-Chip Module Conf.*, Santa Cruz, Feb. 1997, pp. 2-7.

[11] V. Maheshwari, J. Darnauer, J. Ramirez and W. W.-M. Dai, "Design of FPGAs with Area I/O for Field Programmable MCM", *Proc. ACM Symp. on Field-Programmable Gate Arrays*, 1995, pp. 17-23.

[12] M. Marek-Sadowska and S. P. Lin, "Timing Driven Placement", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Nov. 1989, pp. 94-97.

[13] M. Pedram, K. Chaudhary and E. S. Kuh, "I/O Pad Assignment Based on the Circuit Structure", *Proc. IEEE International Conf. on Computer Design*, Cambridge, Oct. 1991, pp. 314-318.

[14] R. V. Raj, N. S. Murty, P. S. Nagendra Rao and L. M. Patnaik, "Effective Heuristics for Timing Driven Constructive Placement", *Proc. Intl. Conf. on VLSI Design*, Bangalore, Jan. 1997, pp. 38-43.

[15] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors: Technology Needs (1997 edition)*, December 1997.

[16] A. Srinivasan, K. Chaudhary and E. S. Kuh, RITUAL: a performance-driven placement algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol.39, (no.11), Nov. 1992, p.825-840.

[17] C. Tan, D. Bouldin and P. Dehkordi, "Design Implementation of Intrinsic Area Array ICs", *Proc. Conf. on Advanced Research in VLSI*, Ann Arbor, Sept. 1997, pp. 82-93.

[18] TimberWolf Systems, Inc., `http://www.twolf.com, 1998.`

[19] R. R. Tummala and E. J. Rymaszewski, eds., *Microelectronic Packaging Handbook*, Van Nostrand Reinhold, 1989.