Floorplanning of Pipelined Array Modules using Sequence Pairs

Matthew Moe Herman Schmit

Carnegie Mellon University Department of Electrical and Computer Engineering

5000 Forbes Ave.

Pittsburgh, PA 15213

{moe,herman}@ece.cmu.edu

ABSTRACT

Floorplanning individual pipelined array modules of a larger overall die can yield beneficial results. Critical paths in every pipeline stage of a pipelined design are roughly equivalent after synthesis. The inability of synthesis tools to predict without full placement both wire congestion and the distance traveled by a wire or wires between consecutive registers are the greatest causes of additional delay and area during place and route. This paper will detail a floorplanning methodology for pipelined arrays that is used to regulate wire congestion and the shortest/longest distances travelled by wire(s) between consecutive registers. A new wire length metric for pipelined arrays will be discussed that attempts to measure the distance travelled by wire(s) between registers. A new move set for floorplanning pipelined arrays using sequence pairs will also be introduced that significantly reduces the annealing design space from previous work. These two contributions when used together have produced up to 10% faster clock periods, 12% smaller designs, and 85% less area used to fix hold time violations in a placed and routed 0.18 μm design.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids - Layout, Placement and Routing

General Terms

Algorithms, Design

Keywords

Floorplan, Pipelined Array, Sequence Pair

1. INTRODUCTION

In digital systems on a chip, where available die area for hardware accelerators is increasing, soft cores are being increasingly used to implement domain specific datapaths. Most of these systems on a chip use many of the same hard cores, i.e. microprocessors, DSPs or memories. What differentiates the various systems are the specialized soft cores which are used to perform or accelerate tasks that are unable to be handled efficiently by other resources on the die. Heavy use of pipelining will be required in order to enable the necessary fast clock speeds in these soft cores.

Pipelined designs send and receive pieces of data at regular intervals. In the general case, registered data can be passed to any pipeline stage from any other pipeline stage while the overall flow of data is from the input to the output. In this work we will restrict the flow of data so that communication is limited to either the previous or next pipeline stage in a strictly systolic data flow. We call this pipelined architecture, which is depicted in Figure 1, a *pipelined array*. Feedback or feedforward paths may be as long as necessary but must be registered in every pipeline stage. Pipelined arrays are commonly used to perform many signal processing tasks.

Areas and clock speeds achievable during synthesis can be unattainable during placement and routing. The greatest causes of additional delay and area during placement and routing are wire congestion and incorrect wire length assumptions during synthesis. In order to alleviate congestion, either wires need to be routed around the congestion or the congested region needs to be expanded. If wires are routed around congested areas, then the delay along that path has been increased, possibly increasing or creating a new critical path in the system. If the wire congested area is expanded then the overall required area for the design will increase.

Synthesis without simultaneous placement and routing relies on the use of stochastic wire length modeling, which can vary significantly from the measured wire length found after place and route. Longer than expected wires may either create setup

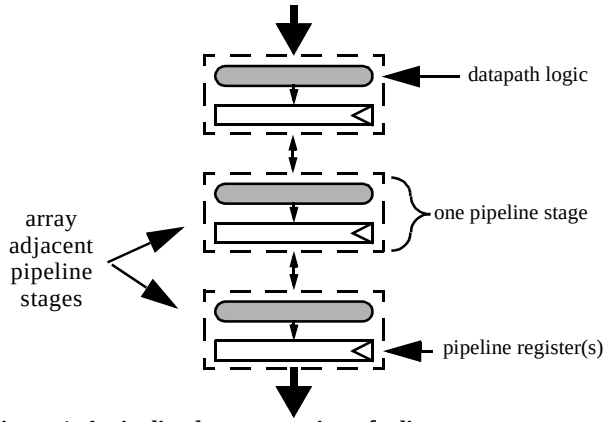

Figure 1. A pipelined array consists of a linear arrangement of logic and registers. One pipeline stage consists of one block of datapath logic and one block of pipeline registers.

ISPD'03, April 6-9, 2003, Monterey, California, USA.

time violations or make a goal clock cycle time unreachable. Shorter than expected wires may create hold time violations. Hold time violations are caused by paths between consecutive registers that are shorter than the clock skew between the two registers plus the hold time of the destination register. One commonly used method to fix this problem is to insert buffers between registers to ensure a minimum delay between registers at the cost of extra area and power. Floorplanning tools should therefore be used to both minimize the longest wire lengths and maximize the shortest wire lengths in order to reduce area usage and increase performance.

This paper will discuss a method of constructing floorplans for pipeline stages of pipelined arrays. These pipelined arrays will only be a portion of the full place and route problem of a chip. A higher level floorplanning tool will still be used to place the entire pipelined array as one module along with other modules.

This work attempts to solve some of the wire length and wire congestion uncertainty of a pipeline array at a high level by focusing on the limited and regular interconnect between pipeline stages. This will reduce the design space explored by a simulated annealing based floorplanning tool to a set where *array adjacent pipeline stages*, which are depicted in Figure 1 as pipeline stages that are adjacent in a pipelined array, are likely to be adjacent in the floorplan. This paper will also introduce a new wire length metric for pipelined arrays that measures the length a single wire or set of wires must traverse between logically consecutive registers in different pipeline stages using what we introduce as the *wire path length*.

2. PREVIOUS WORK

2.1 Data Path Floorplanning

In previous works [1][2] that have attempted data path floorplanning (with simultaneous high-level synthesis), wire delay models were based purely on the center to center distance of modules or the half-perimeter bounding box containing the source and destination modules. Predicting wire length (and thereby delay) can only be accurate when full placement and routing has occurred. Both models will be plagued by inaccurate wire length models because of overly optimistic (center to center) or pessimistic (half-perimeter bounding box) wire predictions.

The work presented in this paper separates the two tasks of synthesis and floorplanning. If the synthesis models and methods are well understood, then a floorplanning tool can be created that understands the shortfalls and strengths of the synthesis and place and route tools. To that end, this floorplanning methodology attempts to limit wire congestion and wire lengths simultaneously.

2.2 Sequence Pairs

Sequence pairs were introduced in [3] as an efficient means of representing every possible nonoverlapping block packing. A sequence pair is composed of two strings (sequences) that together specify directional constraints between every possible pair of blocks. A horizontal constraint is created between two blocks when the order of the two blocks is the same in both sequences. A vertical constraint is created between two blocks when the order of the two blocks in the two sequences differs. Figure 2 graphically depicts these two types of constraints.

The structure created from a sequence pair is apparent when viewed on an oblique constraint graph like the one found in Figure 3. The first sequence is displayed diagonally in the lower left and the second sequence in the lower right. Nodes, which represent floorplan blocks, are located on the constraint graph where matching elements of the two sequences cross. Directional constraints exist between every possible pair of blocks, but only nonredundant edges are drawn on the oblique constraint graph for simplification. A possible floorplan of this graph is also shown in Figure 3.

Directional constraints can be determined from the positioning of nodes alone on the constraint graph. There are four quadrants around every node. The boundaries of the quadrants are the four grid lines that emanate from the node on the graph. Every node that is in the quadrant above the original node is above the block in the floorplan. Therefore constraint edges on an oblique constraint graph show only redundant information. Every block in the shaded quadrants of Figure 4 is above ***B*** in the floorplan.

One sequence pair can lead to an infinite number of floorplans because there are infinitely many possible aspect ratios for every block. Finding the optimal area floorplan while keeping

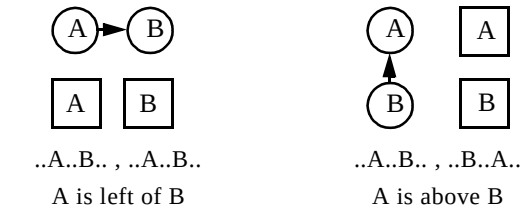

Figure 2. The above figures depict a horizontal constraint (left) and a vertical constraint (right). Dots in the sequence pairs represent possible locations of other blocks.

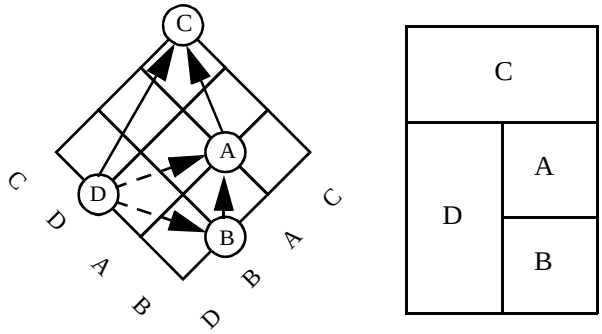

Figure 3. Oblique constraint graph (left) and possible floorplan (right) for the sequence pair CDAB,DBAC. Dashed/Solid lines represent horizontal/vertical constraints.

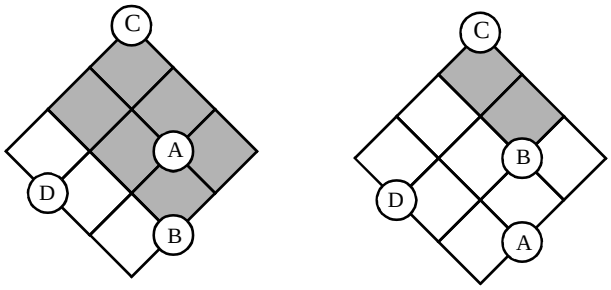

Figure 4. The shaded areas in both oblique graphs denote the area where blocks that are above B are found.

the aspect ratio of the blocks within a given range is a constrained optimization problem and is NP-Complete [4]. The solver used throughout this work can be found at [9]. Many other works have shown possible solutions to this problem. [5] uses a branch and bound algorithm which becomes impractical as the number of blocks increases. [6] uses constrained optimization [7] to create a block placement. A faster solver using Lagrangian Relaxation can be found in [8].

Simulated annealing of hard block floorplans with sequence pairs was first described in the original paper on sequence pairs [3]. The move set consisted of randomly swapping elements in either the first or second sequence or rotating one of the elements. Later work [6] added soft and preplaced blocks to this methodology. The simulated annealing cost functions for both methodologies were based on total area and a wirelength metric. The location of input and output ports are known when using hard blocks which enables more accurate modelling of wire lengths. When using soft blocks, wire length models are only estimations.

3. FOPA - FLOORPLANNING OF PIPELINED ARRAY MODULES

Unlike previous work on sequence pairs, this work targets only a small portion of the overall chip floorplanning problem - floorplanning of pipelined array modules. Specialization allows for a number of changes to the general sequence pair framework that significantly reduces the design space and improves the quality of results from a more general floorplanning optimization tool. The following sections describe our additions to the general simulated annealing algorithms for sequence pairs.

3.1 Block Partitioning

Pipeline registers are heavily used in a pipelined design to reduce the critical path delay within a single pipeline stage and thereby increase data throughput. While logically placing registers, a designer faces the dual goals of balancing delay between pipeline stages and minimizing the number of registers in order to minimize area. Pipelined registers are therefore placed, whenever possible, where the fewest number of data bits need to be registered. Partitioning a floorplan after pipeline registers will also limit the number of connections that pass between floorplanning blocks. In this work, blocks are partitioned immediately after the pipeline registers - one floorplanning block is composed of one pipeline stage.

It is possible to generalize our approach by allowing a floorplanning block to consist of ***N*** pipeline stages. This would reduce the total number of blocks during floorplanning and reduce the complexity and runtime of the floorplanning tool. Place and route results are likely to worsen as the granularity of the floorplanning blocks decreases.

3.2 Oblique Connectivity Graph

An *oblique connectivity graph* is a representation that depicts both wire connectivity and block placement simultaneously. The graph consists of an oblique graph with edges that represent wires between array adjacent blocks instead of directional constraints. Figure 5 shows two such graphs.

Edges on an oblique connectivity graph for a pipelined array can represent many wires. Routing multiple connections between blocks over other blocks will probably create routing congestion which will ultimately cause extra area usage or decreased performance. These problems can be ameliorated by constraining the simulated annealing design space to a set where array adjacent stages are adjacent in the floorplan. In a sequence pair this can be accomplished by not allowing the same block between any two array adjacent blocks (stages) in both sequences. In an oblique connectivity graph, *array illegal* sequence pairs have nodes that can be found inside a tilted rectangle bounded by array adjacent blocks at the corners. In Figure 5, the graph on the left has a shaded rectangle demarcated by the blocks ***B*** and ***C***. This is an array illegal representation because block ***A*** is inside this rectangle. The representation on the right is array legal because there are no blocks inside the rectangle bounded by blocks ***B*** and ***C*** or any other pair of array adjacent blocks. Figure 6 has psuedo-code for determining array legality.

It is still possible for an array legal representation to have wires that need to be routed over other blocks. The number of reachable positions with this characteristic has been severely reduced by only allowing array legal representations as can be seen in Table 1. Every array illegal position will have wires that must be routed over another block. The least wire congested floorplans can therefore be found within the array

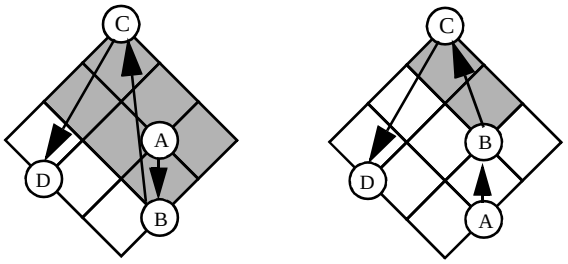

Figure 5. Edges on an oblique connectivity graph denote connections between blocks. The graph on the left is array illegal because block *A* lies in a box bounded by B and C. The right graph is array legal.

```
graph_is_legal = TRUE;
for(i=1;i<n;i++){  // n = number of pipeline stages
  start_index_1 = position of i in sequence 1
  end_index_1 = position of (i+1) in sequence 1
  start_index_2 = position of i in sequence 2
  end_index_2 = position of (i+1) in sequence 2
  for(j=start_index_1+1;j<end_index_1;j++(or j--)){
    for(k=start_index_2+1;k<end_index_2;k++(or k--)){
      if (if block at position j in sequence 1 ==
        block at position k in sequence 2){
      graph_is_legal = FALSE; }}}}
```

Figure 6. Psuedo-code for determining array legality.

Table 1. Assuming *n* blocks, there are *n*! * *n*! sequence pairs. There are fewer array legal sequence pairs.

n	*n*! * *n*!	legal sequence pairs
3	36	28
4	576	284
5	14,400	3,860
6	518,400	66,892
7	25,401,600	1,422,348

legal design space. Wire congestion is one of the greatest causes of increased delay during placement due to longer than expected wire routes. Reducing wire congestion can also reduce the total area required for the design during placement and routing.

3.3 Wire Path Length Model

Every wire length within or even between blocks cannot be accurately measured during soft block floorplanning. Classic wirelength models are well suited for measuring individual wires in random logic designs. *Wire path length* more accurately reflects the wiring situation of a pipelined array. This model measures the likely distance a wire or set of wires must travel from one register to the next while also reaching all points within the destination register's block.

Using the longest wire path length as a metric allows a component of total delay (wire length) to be considered during simulated annealing but not as a measurable quantity. After synthesis, the delay across all pipeline stages is roughly equivalent. During placement and routing, unforeseen additional wire capacitance and congestion are the greatest causes of additional delay. The wire path length model attempts to proportionally model to first order the capacitance likely to be seen on wire paths between logically consecutive registers. The only way to accurately model every wire in a placement is by placing every hard block whether that be a microprocessor or a standard cell and routing the design.

Ideally, a placement tool will place the registers within a pipeline stage closest to the block that contains the next pipeline stage. In our model, all the registers for a block are assumed to be located on the block edge closest to the next block in the pipelined array. The wire path length model measures the likely distance traveled by a wire or set of wires between the worst case possible placement of registers.

Figure 7 contains psuedo-code for the measurement of wire path length. Figure 8 demonstrates the measurement of wire path length for block ***B*** in two examples. The hatched region depicts the mostly likely location of registers within a block. If the registers are located on the side of the block adjacent to the hatched region, then the wire path length for block ***B*** is width(***B***) + height(***B***) for the floorplan on the left and (2 * width(***B***)) + height(***B***) for the floorplan on the right. This added width will discourage the use of this arrangement during simulated annealing which is more likely to have wire congestion and longer wire lengths (and wire delay) than the floorplan on the left. Previous wire length metrics would not have discouraged this arrangement as much. Finding the longest wire path length in a floorplan is an *O(n)* operation.

```
max_wpl = 0;  // maximum wire path length
for(every set of array adjacent blocks (a,b,c)){
 if( a|b )
  current_wpl = x[b]+w[b]-(x[a]+w[a])
     +max(y[a]+h[a]-y[b],y[b]+h[b]-y[a]);
 if( a / c b )
  current_wpl += w[b];
 if( a b / c )
  current_wpl+=max(0,y[b]+h[b]-(y[c]+h[c]));
 max_wpl = max(max_wpl,current_wpl);
}
```

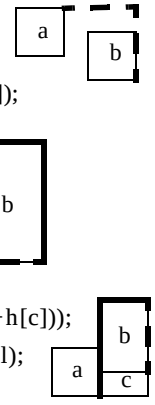

Figure 7. Pseudo-code for calculating Wire Path Length. Other cases are necessary but are flipped or rotated versions of the above cases.

3.4 Annealing Cost Metric

The cost function used for our simulated annealing algorithm is: $\frac{Area}{MinArea} + \frac{MaxWPL}{MinWPL} + max\left(\frac{TarAR}{AR}, \frac{AR}{TarAR}\right)$ This measures the sum of ratios difference the current floorplan is from an ideal floorplan. *Area* is the total area used by a floorplan including dead space. *MinArea* is the synthesized area. *MaxWPL* is the length of the longest wire path length found in a floorplan. *MinWPL* is the ideal length of the shortest possible maximum wire path length for a floorplan. *AR* is the aspect ratio of the current floorplan. *TarAR* is the target aspect ratio of the final floorplan. *MinArea*, *MinWPL*, and *TarAR* are constants for a particular design. *Area*, *MaxWPL*, and *AR* are the metrics determined from the design during annealing.

The ideal length of the shortest possible maximum wire path length for a floorplan is twice the square root of the area of the largest block. This can be found in practice when the largest block is square, the previous pipeline stage is adjacent to the largest block and the next block is not on the same side of the largest block as the previous block.

3.5 Move Set

The movement of elements within our modified sequence pair framework consists of the removal of one element in one of the sequence pairs and the reinsertion of that element into an array legal location in the same sequence pair. The element to be moved and the sequence pair to be modified are both randomly chosen. The span of array legal destination locations are all contiguous and are bounded by either the ends of the sequence pair or an array illegal destination. An *array illegal destination* is one where the moved element would be placed between two array adjacent elements in the pipelined array. Moving the element past one of these array illegal destinations will always create array illegal positions. One such position which is difficult to find (and is not found with our psuedo-code) is when two sets of array adjacent blocks have wires that cross each other. This position will never be reached if starting from a legal position because it requires movement of an element

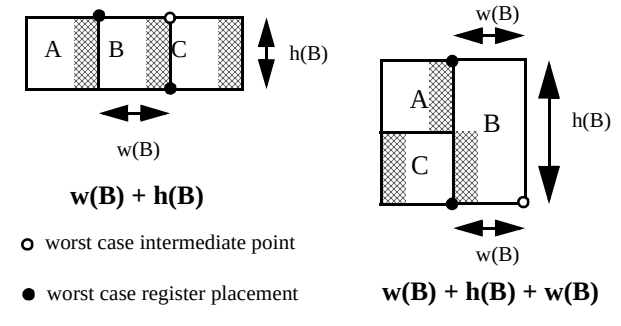

Figure 8. Measurement of wire path length for block *B* in two different floorplans. The hatched regions denote the most likely location of registers.

past an array illegal destination. Figure 9 shows some array legal and array illegal moves. Every array legal move is reversible.

3.6 Move Set Completeness

In order for a search algorithm like simulated annealing to work, it is necessary to show that the restricted move set previously introduced is complete - that every array legal design point in the design space is reachable from any other array legal design point. Due to space limitations, this proof is not contained in this paper.

4. EXPERIMENTS AND RESULTS

This section will discuss experiments and results using different simulated annealing floorplanners with our new wire path length metric and other wire length metrics. The floorplanners will differ in the move set used and the initial starting position. Floorplanning results with our new restricted move set will be shown to be better than an unrestricted move set. More importantly, place and route results will show that our new restricted move set creates designs up to 10% faster, 12% smaller, or use 85% less area to fix hold time violations in a placed and routed 0.18 µm design compared to an unfloorplanned design. The results will also show that the FoPA move set is better than previous floorplan methodologies for floorplanning pipelined array modules. Because this work concentrates on module floorplanning rather than chip floorplanning, ultimately the methodologies are compared using complete and full place and route results.

Floorplanning tools are currently used to break down a large place and route task into many smaller tasks. Individual modules are then placed and routed separately. In the FoPA methodology and for all the results in this paper, the floorplanners were used to create placement constraints in the place and route tool. All pipeline stages were placed and routed simultaneously in one tool.

4.1 Methodology

There are five different simulated annealing module floorplanners used throughout the rest of this paper. ***Classic*** is used to denote a simulated annealing floorplanner with an unrestricted swap move set and a random initial placement. ***Classic+LSP*** denotes a simulated annealing floorplanner with an unrestricted swapping move set and an array legal starting position. ***FoPA*** denotes a simulated annealing floorplanner with our new restricted deletion/insertion array legal move set and an array legal starting placement. All three of these floorplanners use wire path length as their wire length metric. ***FoPA+BB*** and ***FoPA+Center*** are both identical to ***FoPA*** except for the use of the half perimeter bounding box or center to center wire length metric. All five floorplanners were built upon a simulated annealer using sequence pairs found at [9].

The array legal initial placement used for the ***Classic+LSP*** and ***FoPA*** simulated annealers is a straight line arrangement of the pipeline stages in the floorplan. This poor but legal starting position was chosen because randomly generating an array legal starting position is increasingly difficult as the number of blocks increases. The ***FoPA*** floorplanners are the only floorplanners that maintain array legality at all times during annealing. The ***Classic*** floorplanners allow array illegal moves without penalty.

Constraints on aspect ratios for blocks within a floorplan are set to ensure that the longest possible wire path length through the blocks is identical if all of the blocks are arranged in a line. This means that the smaller blocks are given the widest latitude of aspect ratio ranges and vice versa. In order to allow some aspect ratio range in even the largest block, the largest possible block contribution to wire path length is allowed to be at most 10% more than twice the square root of the largest block. Other factors, such as the distance between blocks, can still lengthen the wire path length beyond 10%. Allowing 10% aspect ratio flexibility lets the largest block have an aspect ratio range of between 0.642 and 1.56 while changing the maximum contribution of the block to wire path length by less than 10%. The wider the aspect ratio ranges are, the more likely that a dense floorplan will be found.

The initial temperature of all the annealers is set to be the initial cost. This ensures an initial acceptance rate of approximately 95% or higher. At each temperature, 100 * (the number of blocks) moves are attempted. The cooling schedule is $T(i) = 0.9 * T(i\text{-}1)$. The placement is considered cold when the cost is unchanged for 3 consecutive temperatures.

All placed and routed designs are in a 0.18µm design process.

4.2 Wire Path Length Design Space

Three designs were annealed 50 times to create the three design space plots of Figure 10. Two of the designs, a 43 block design and a 60 block design, implement one round of IDEA encryption [10]. The final design is composed of the first 20 blocks of the 43 block design. The three designs have block sizes that differ by as much as 4x. The 20 block design space shows that the ***FoPA*** floorplanner performs as well as the ***Classic*** floorplanner in terms of area while outperforming the ***Classic*** floorplanner in terms of wire path length. The 43 block design space and the 60 block design space show that as the number of blocks increases, the ***FoPA*** floorplanner well

CDBA DABC

Legal Sequence 1 Moves	Legal Sequence 2 Moves	Illegal Sequence 1 Moves	Illegal Sequence 2 Moves
A,B - CDAB DABC	A,D - CDBA ADBC	A - **C*A*DB D*A*BC**	B - **C*D*BA B*D*AC**
C,D - DCBA DABC	A,B - CDBA DBAC	B,D - **C*B*DA D*A*B*C***	C - **C*D*BA C*D*AB**
	A - CDBA DBCA	C - **D*BC*A D*A*BC**	D - **C*D*BA AB*D*C**
	B,C - CDBA DACB		
	C - CDBA DCAB		

Figure 9. All array legal and bounding array illegal moves for the sequence pair CDBA DABC. To the left of each sequence pair is the element that was moved to produce the new sequence pair. Moving an element past an array illegal move is disallowed because either the resulting move is illegal or the resulting floorplan will have crossing wire paths. The cause of the positions being illegal are showcased in bold. Elements in italics denote the element that is between two connecting elements of a pipelined array.

outperforms the two ***Classic*** floorplanners in terms of wire path length while performing as well as the two ***Classic*** floorplanners in terms of area. Runtime for the ***FoPA*** floorplanner was on average about half the runtime of the two ***Classic*** floorplanners.

4.3 Synthesis to Full Place and Route

Five designs have been synthesized using Synopsys' Design Compiler [12] and placed and routed using Monterey Design Systems' Dolphin [13] using the best floorplans returned from 10 simulated annealing runs of the five different annealing floorplanners, as well as an unfloorplanned design. The five implemented designs are a 1-D DCT (composed of 12 blocks), a 2-D DCT (32 blocks), a 43 pipeline stage IDEA round (43 blocks), a 60 pipeline stage IDEA round (60 blocks), and a small Low Density Parity Check decoder [11] (27 blocks). Synthesized area of the various designs can be found in Table 2. Results from the simulated annealing runs can be found in Figure 11. Floorplan utilization measures how much of the total area is used by floorplan blocks. As can be seen from these results, the ***FoPA*** floorplanner outperforms the ***Classic*** floorplanners in terms of wire path length while performing as well as the ***Classic*** floorplanners in terms of

Table 2. Area achievable during synthesis

Application	Synthesized Area (μm^2)
1-D DCT	668,323
IDEA(short)	948,461
IDEA(long)	1,082,773
2-D DCT	1,505,284
LDPC	4,508,761

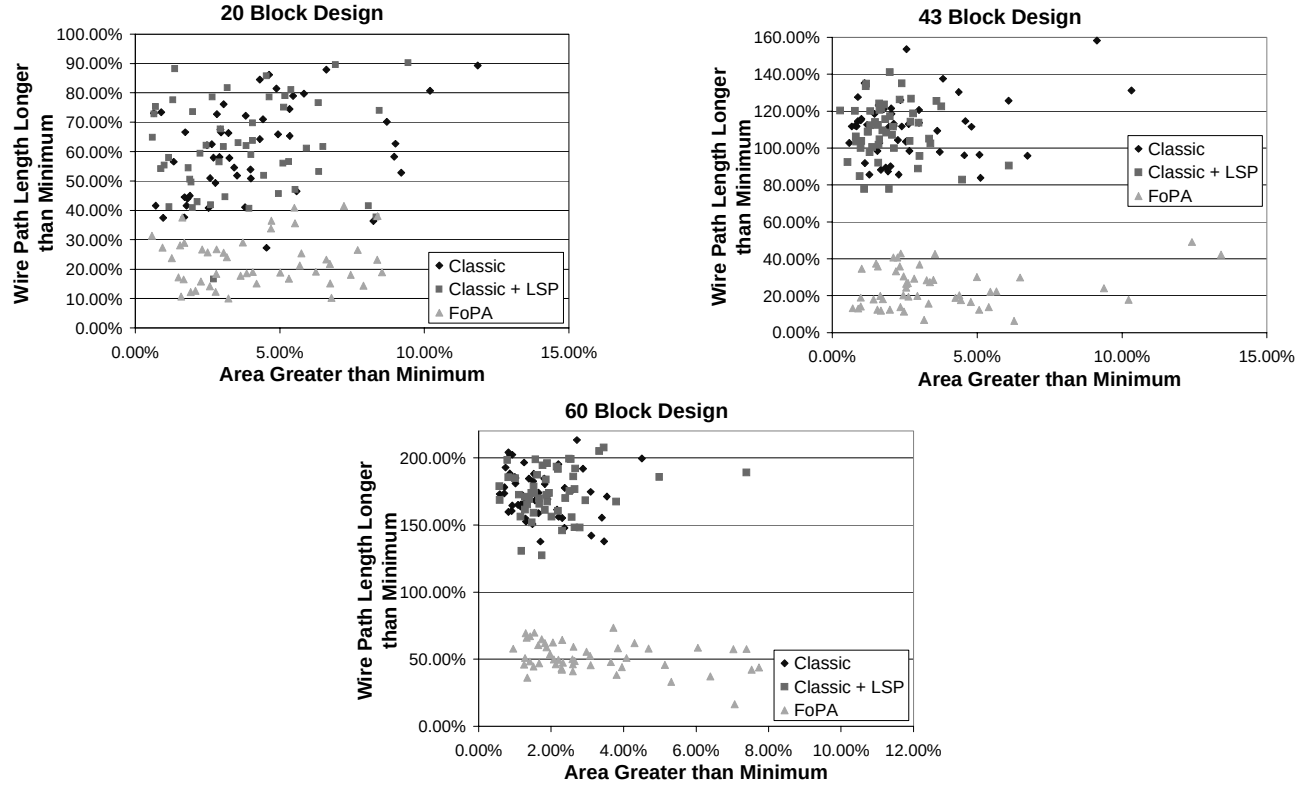

Figure 10. The FoPA move set produces better wire path lengths than the classic methodologies as the number of blocks increases.

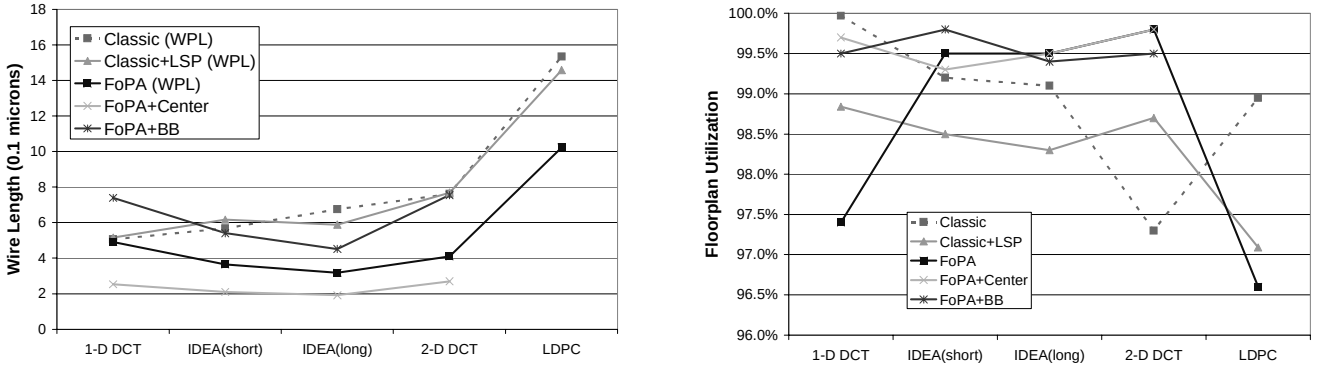

Figure 11. Best Wire Length and Floorplan Utilization found after 10 runs of each annealer.

utilization. Two floorplans for the 1-DCT found using the ***Classic*** and ***FoPA*** floorplanners can be found in Figure 12. The three floorplanners that use wire path length created 1-DCT designs that are very similar in wire path length due to the small number of blocks (12) and small design space to be explored. The wire lengths of the floorplanners that do not use wire path length are included for completeness in Figure 11, but these results are not directly comparable to each other or to the floorplanners that use wire path length.

All of the designs were unable to be placed and routed with their initial floorplans without Design Rule Errors. The floorplan sizes were increased in increments of 1% until a design without Design Rule Errors was found.

Figure 13 shows the total area required for a placed and routed design compared to the ***FoPA*** methodology. Figure 14 shows both the difference in dead space between the pre-placed and post-placed dead spaces and the area that is used to fix hold time violations as a percentage of synthesized area. The difference in dead space is caused by the addition of buffers to fix hold time violations and the resizing of gates to meet clock cycle constraints. Between 7% and 9% of the synthesized area added during place and route is due to resizing. The rest of the increase in area can be attributed to fixing hold time violations. As can been seen from the results for most of the applications, required hold time area is much higher in the unfloorplanned than the floorplanned designs. This is probably caused by the fact that the place and route tool attempts to have logically connected registers as close together as possible. Taken to an extreme, this can easily create shorter and thus faster paths than the clock skew plus hold time on registers in the design. Floorplanned designs tend to both limit the minimum as well as the maximum separation between logically connected registers. Hold time violation fixing area not only increases the area used on a chip but also increases the power consumption.

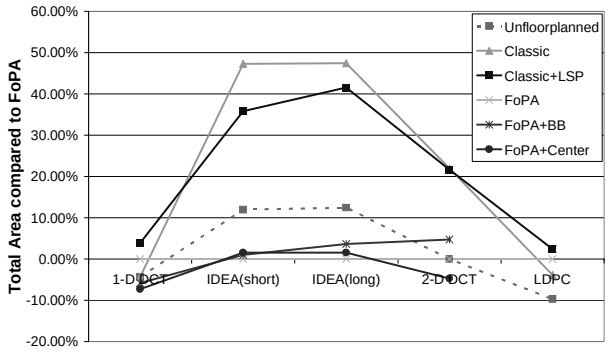

Figure 13. Total area used by designs after place and route compared to the FoPA methodology.

LDPC has virtually no area used to fix hold time violations. There are very few direct register to register connections in this application. Most of the paths between registers have logic between them thereby reducing the possibility of hold time violations. 1-D DCT and 2-D DCT are virtually the same application. If purely placed back to back then the hold time area percentages would probably be identical. In between the two 1-D DCT designs in the 2-D DCT is a set of transposition registers. These are all purely register to register connections thereby raising the possibility of hold time violations. The two IDEA encryption applications also require numerous register to register connections. These results show that if there are many register to register connections, then the ***FoPA***

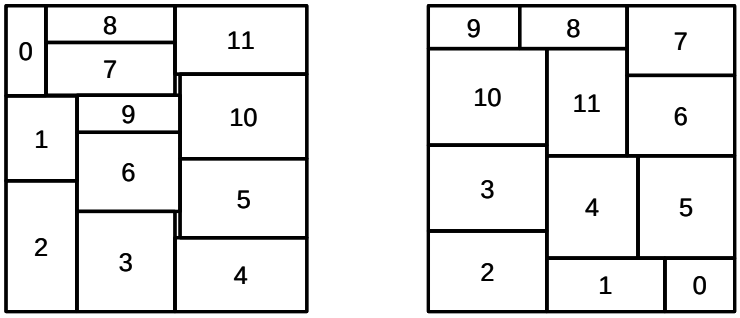

Figure 12. Best floorplans for 1-DCT using *Classic* (left) and *FoPA* (right) methodology. Notice that wire congestion will probably be caused around blocks 6, 7, 8, and 9 in the *Classic* floorplan.

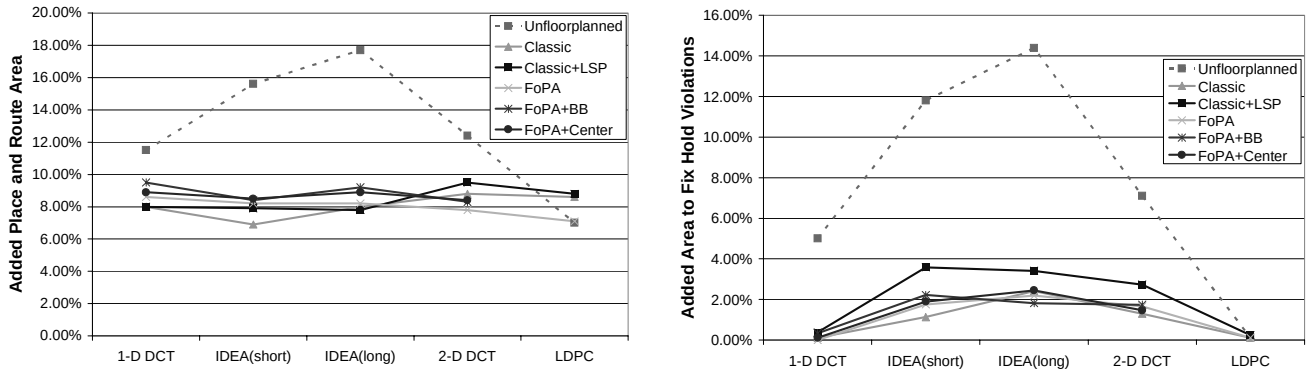

Figure 14. Difference in pre- and post-place and route dead space as a percentage of synthesized area and the Hold Area used to fix hold time violations during place and route as a percentage of synthesized area.

methodology will use less area than an unfloorplanned design or other floorplan methodologies.

Table 3 shows the achievable clock periods of the various designs after synthesis. Figure 15 shows the percentage by which the clock periods are missed after place and route. As can be seen from the results, the ***FoPA*** methodology is always the fastest or very close to the fastest design. The ***FoPA*** floorplanner creates as fast or, as is usually the case, faster placed and routed designs than the unfloorplanned design. The maximum wire path length found during simulated annealing predicts quite accurately which of the floorplan methodologies will produce the fastest result. When the maximum wire path lengths are nearly identical after annealing as can be found in the 1-DCT results, the achievable clock speeds are nearly identical also, with the shortest wire path length being the fastest design and vice versa.

Maximum wire path length is not a perfect or proportional predictor of performance. Wire delay is still just a fraction of total delay. Attempting to decrease or limit wire delay can only improve the performance by as much as wire delay limits the performance of the design.

5. CONCLUSIONS

Currently, blocks of the size used in this paper are unfloorplanned. We have shown that a floorplanned design uses less area to fix hold time violations than an unfloorplanned design.

A new floorplanning move set has been shown that reduces the simulated annealing design space to a set where array adjacent elements are adjacent in the floorplan. This produces designs that are faster and require less total area than previous floorplanning methodologies due to reduced wire congestion.

Table 3. Clock Periods achievable during synthesis

Application	Synthesized Clock (ns.)
1-D DCT	2.13
IDEA(short)	1.88
IDEA(long)	1.90
2-D DCT	2.13
LDPC	3.55

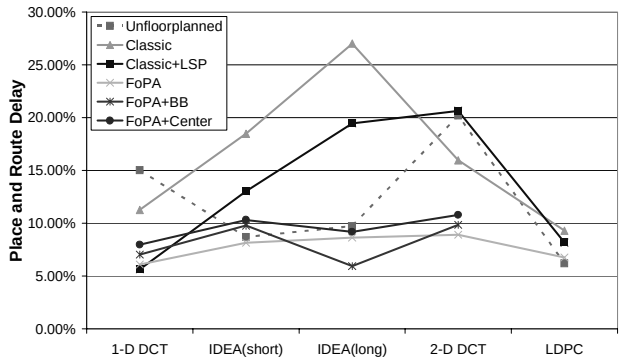

Figure 15. Clock periods achievable during synthesis and the extra delay added during Place and Route as a percentage of synthesized clock period.

A new metric for wire path length has been shown that attempts to measure the worst case distance a wire or set of wires must travel between registers in consecutive pipeline stages. This metric more accurately reflects the wiring of pipelined arrays than previous wire length metrics for random designs. Compared to previous wire length metrics, wire path length has been shown to usually produce faster designs.

The restricted move set is used to reduce wire congestion while the wire path length metric is used to reduce wire delay. These two contributions when used together have been shown to produce up to 10% faster clock periods, 12% smaller designs, or use 85% less area to fix hold time violations than an unfloorplanned design.

6. ACKNOWLEDGMENTS

This project was supported in part by the Microelectronic Advanced Research Corporation (MARCO) under Contract No. 98-DT-660. Professor Schmit was supported in part by a CAREER grant from the National Science Foundation. The authors wish to thank ST Microelectronics for their technical support.

7. REFERENCES

[1]J. Crenshaw, M. Sarrafzadeh, P. Banerjee, P. Prabhakaran, "An incremental floorplanner," in *Proc. Ninth Great Lakes Symposium on VLSI*, pp. 248-251, 1999.

[2]Y.M. Fang, D.F. Wong, "Simultaneous Functional-Unit Binding and Floorplanning", in Proc. ICCAD, pps. 317-321, 1994.

[3]H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "VLSI Module Placement Based on Rectangle Packing by the Sequence Pair," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518-1524, December 1996.

[4]L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," Inform. Contr., vol. 57, pp. 91-101, 1983.

[5]S. Wimer, I. Koren, I. Cederbai,. "Optimal Aspect Ratios of Building Blocks in VLSI," *IEEE Trans. on Computer Aided Design*, vol. 8, no. 2, pp. 139-145, December 1996.

[6]H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/preplaced modules," in *Proc. Int. Symp. Physical Design*, Apr . 1998. pp. 167-172.

[7]P. M. Vaidya, "A New Algorithm for minimizing convex functions over convex sets," *Mathematical Programming*, 73:291-341, 1996.

[8]F. Y. Young, C. C. N. Chu, W.S. Luk, and Y. C. Wong, "Handling Soft Modules in General Nonslicing Floorplan Using Lagrangian Relaxation," in *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, pp. 687-692, May 2001.

[9]http://www.cse.ucsc.edu/research/surf/GSRC/progress.html#V

[10]Bruce Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, Inc., 1996, pp. 319-325.

[11]R. G. Gallager "Low Density Parity Check Nodes," *IRE Trans. Inform. Theory*, Jan. 1962, pp. 21-28.

[12]"Design Compiler Reference Manual", *Synopsys Inc.*, 1999.

[13]"Dolphin 2.0 Online Help", *Monterey Design Systems*, December 2000.