Transformational Placement and Synthesis

Wilm Donath,Prabhakar Kudva,Leon Stok
IBM TJ Watson Research Center
Yorktown Heights

Paul Villarrubia
IBM Server Group
Austin, TX

Lakshmi Reddy,Andrew Sullivan,
Kanad Chakraborty
IBM Server Group
Hopewell Junction, NY

Abstract

Novel methodology and algorithms to seamlessly integrate logic synthesis and physical placement through a transformational approach are presented.

Contrary to most placement algorithms that minimize a global cost function based on an abstract representation of the design, we decomposed the placement function into a set of transforms and coupled them directly with incremental timing, noise, and/or power analyzers. This coupling results in a direct and more accurate feedback on optimizations for placement actions.

These placement transforms are then integrated with traditional logic synthesis transforms leading to a converging set of optimizations based on the concurrent manipulation of boolean, electrical, as well as physical data.

Experimental results indicate that the proposed approach creates an efficient converging design flow that eliminates placement and synthesis iteration. It results in timing improvements, and maintains other global placement measures such as wire congestion and wire length.

The flexibility of the transformational approach allows us to easily add, extend and support more sophisticated algorithms that involve critical as well as non-critical regions and target a variety of metrics including noise, yield and manufacturability.

1 Introduction and Related Work

Timing optimization in traditional logic synthesis is based on a transformational approach. The net list is gradually modified and refined. Timing, noise and power analyzers incrementally measure the design and provide feedback to the transforms that make the actual design changes [23]. An evaluator (or the transform itself) queries the analyzers and decides if the design actually improves and accepts/rejects the netlist modifications.

The advantage of the above approach is that direct feedback from the analyzer(s) is used in the synthesis optimizations. There is a direct coupling between the analyzers used for the final sign-off criteria and the optimization. This direct coupling allows discrete logic and electrical netlist optimizations within synthesis.

Algorithms for placement have the advantage of a rigid underlying mathematical formulation. They have been very successful in optimizing net length and controlling wire congestion and their complexities scale well to handle larger designs. Most placement algorithms use continuous formulations and hence do not lend themselves to discrete optimizations typically used in synthesis.

Timing driven placement techniques have often used the ability to specify constraints into the placement algorithm such as net weights and capacitance targets to achieve such goals [9, 20, 14]. However, they do not directly take into account feedback from for example a timing analyzer. These techniques formulate their problems as continuous optimization problems and hence do not lend themselves easily to include netlist transformations which are discrete in nature.

Including these objectives directly in the problem formulation leads to expensive optimization algorithms. In [21], locations are specified as variables for timing improvement and an exact non-linear optimization problem is formulated to achieve this goal. However, the run time of non-linear methods tends to grow quickly with the size of the designs.

A primary approach has been to use a snapshot of placement as a starting point for netlist transformations, followed by an incremental placement step [12, 18, 16, 17] to legalize the perturbations caused by the netlist transforms. These approaches significantly limit the netlist changes that can be made to be able to maintain incrementality in the succeeding placement. In POINT [22], the approach is extended by adding a flow-based placement improvement phase as a legalization step, thereby increasing the number and scope of network changes that can be tolerated. In [11], a methodology that enables one to invoke synthesis transforms in the intermediate steps of a partitioning based placer is described.

All these approaches start from an existing placement and only try to optimize around this initial local solution. In our transformational placement and synthesis approach (TPS) we take this a step further. By creating a sequence of more and less granular placement and netlist modification transforms a converging design closure process is created, starting from just a netlist without initial placement. The placement function is decomposed into a set of placement trans-

forms addressing each a specific phase of the placement problem. Each placement step becomes just another transform that changes the design space, in this case the placement of cells. These placement transforms can be freely mixed and matched with the traditional logic synthesis transforms that change the netlist. The accuracy versus runtime tradeoff of these optimizations can be refined as the quality of the placement and netlist data improves in a converging flow.

All transforms have an unified view of the placement and synthesis design space. Synthesis, timing, and placement algorithms and data are concurrently available to all transforms. This opens up the possibility for an entire new class of transforms that modify the netlist and placement concurrently. A transform to eliminate wire congestion can do this both by moving cells or re-decomposing a piece of the netlist. An electrical correction transform can let its choice to clone a cell or buffer its output be driven by how much space is available to do one or the other. Section 4 describes various of these transforms.

To be able to apply this transformational approach to industrial size designs, efficient data structures need to be employed. Section 2 describes our abstraction of the placement image. This image gradually refines itself during the course of the algorithms thereby providing efficiency up-front and precision in the final stages of the design flow. All timing calculations in TPS are fully incremental and recalculations only happen in regions affected by netlist or placement changes. Section 3 describes how the wire-length (and load) calculation is done. Section 4.1 describes how the placement problem is decomposed such that we maintain the global properties (like wire-length and wire congestion) and in addition can address local problems (like timing and noise).

Section 5 describes how the placement, synthesis and combined netlist/placement transforms can be grouped in scenarios to obtain an efficient (order of runtime similar to sum of synthesis and placement stand-alone) and converging design flow.

2 Placement Image

The placement data of a design needs to be represented at a variable level of abstraction. On one hand the cell placement needs to be represented precise enough to get accurate information from the analyzers. On the other hand one does not want to spend a lot of time updating detailed placement information that does not have a major impact on the analysis results.

A bin-based placement image is selected since it can be efficiently updated and can gradually represent more precise placement information. The chip/design area is divided into *bins* as shown in figure 1. Only abstracted information is maintained with respect to each bin. Each bin has associated with it a certain cell capacity, and wiring capacity. Circuits can be moved between the bins without a complex legalization procedure. Instead, we keep track of a simpler measure of how much of the *bin capacity* is used up by circuits already placed in the bin.

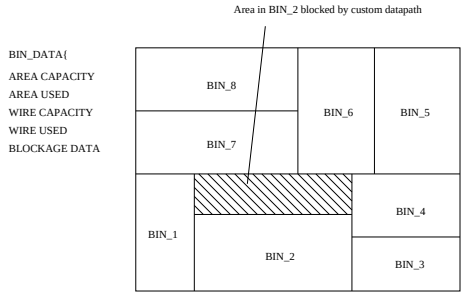

Figure 1. Coarse Image View

The bin structure has functions that relate to the physical characteristics of the chip image: where and how many circuit locations are available, where i/o's are placed, large partitions are placed, block space for other circuits, where power lines are placed and how they block other wiring. This information is sufficient to ensure that a legal detailed placement can be obtained and that the wire-ability metrics for the routing are met.

The bins can have any size. The smaller the bin, the more precise the placement of the cells in this bin. Eventually, each bin could contain one cell and the cell will be fixed in the location of the bin. In the case of detailed locations, the circuits have exact legal locations for a given chip image and the circuit rows and wiring tracks are exactly defined.

The bin structure is especially beneficial in a synthesis/physical design environment where significant changes are made to the design and maintaining legal locations for detailed placement would be expensive. The bin structure naturally supports our optimization flow, where more drastic restructuring decisions are made up-front, and smaller decisions, supported by more precise analysis information later. Gradual refinement of the bins will create gradually more precise wire-length estimates and better timing and noise analysis.

3 Wire Length Calculation

Figure 2 shows the tracking between the net-length of a Steiner tree and the final routing of the net. It shows the number of nets that have a certain percentage prediction error. The three data sets shown (left to right) were obtained by successively removing the shortest 10% and 20% of nets from the statistics. One can see that all larger error percentages disappear if the shortest nets are removed. The error due to short nets does not have a significant effect on delay. For the slightly longer and long nets the precision of a Steiner length approximation is sufficient for the transformations that are done in the TPS phase. This is especially true, if this Steiner tree is also being used to initialize the global router. Also a final sizing after routing can be done to compensate

for the in-accuracies of the Steiner tree without changing the placement.

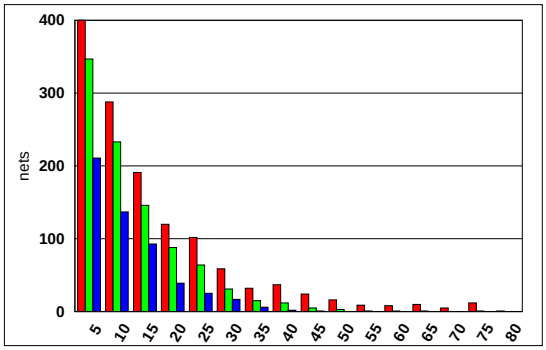

Figure 2. Wire load histogram

The Steiner tree is calculated using the positions of each cell and its pins. Cell positions might be either exact (in the final phases) or derived from the bins the gate is in. The Steiner tree gets dynamically re-calculated when gate positions change as well as when new cells are created or old ones deleted.

For short wires, an Elmore delay [25] model is used. The wire load capacitances are estimated as lumped capacitances proportional to the Steiner estimates of the lengths of the wires. For longer wires where the RC component is significant, an appropriate delay model [19, 5] is chosen.

These models are registered as net-delay calculators in an incremental timing analysis engine [10]. Both changes to positions of cells and changes to the netlist may trigger incremental recalculations of the timing and Steiner trees. We can have different wire-length calculators for wires within the bins. For example, one may use approximate wire lengths obtained from Rent rule [6, 7].

4 Transformations

4.1 Placement Transformations

Different placement algorithms need to be applied at different stages of the design flow. The placement algorithms deployed within TPS include multi-level partitioning [2, 13], look ahead min-cut [4], clustering, reflow, quadratic [14], and collections of greedy detailed placement heuristics. Most of these placement algorithms were developed and tuned within a placement tool [11]. The dominant flow during most placement transforms is that of the traditional bipartitioning placement methodology. In this flow, the starting point is a netlist consisting of fixed and movable circuits, along with floorplanning constraints such as primary IO port assignments, reserved areas, capacitance targets, and areas assigned for a special set of logic circuits.

The *Partitioner* transform separates the movable objects into two partitions defined by a dividing line (cut line) that

Algorithm Partitioner+Reflow

```
/* Initialize the window list to one window that covers the entire design. */
start_head = initialize_window_list();
for(cut_number = 0; cut_number < number_of_cuts; cut_number++)
      /* Partitioner */
      for(window = start_head; window != NULL; window = window->next)
         /* extract the contents of this window from the chip */
         this_subset = get_subset(window);
         /* partition this subset using multi-level partitioning */
         partition_subset(this_subset);
         /* Put the partitioned results back into the chip */
         put_subset(this_subset);
      end for;
      /* Reflow */
      start_head = subdivide_each_window_in_the_list(start_head);
      for(window = start_head; window != NULL; window = window->next)
         reflow_window = merge(window,window->next);
         this_subset = get_subset(reflow_window);
         partition_subset(this_subset);
         put_subset(reflow_window);
      end for;
end for;
```

splits the design into two halves. The objects for each partition are selected so as to minimize the total number of wires that cross the cut line, and so that the amount of circuit area and connections in each partition are approximately equal. At this point we have two partitions. Applying the same algorithm recursively, the two partitions become four, then eight and so on. For every partition operation following the first there is the added difficulty of terminal projection, which is the method for modeling connections that flow from one partition into another. TPS handles terminal projection by making the entire netlist of objects, and their placement locations visible to every partitioning operation. In this way there is no data model set up overhead. Connections that exit a partition are seen natively, and reacted to accordingly.

One of the criticisms of the partitioner transform is that decisions made early on in the process tend to ”trap” objects into certain geometric areas, providing little means for them to escape what may have been a suboptimal assignment. A *reflow* algorithm can resolve these situations. Logic is given the opportunity to flow back into areas that the strict bipartitioner has excluded. This is accomplished by deploying a series of sliding windows that roam around the chip in between the partitioning steps of the bipartitioner. In this process the size of the roaming windows start off large and progress to small in size. In addition to the reflow transform, other techniques such as varying the initial cut line coordinates and using detailed placement techniques also help. With these algorithms deployed the resulting placement distributions visually appear less grainy than strict bipartitioning, and look similar to the results of a simulated annealer. The partitioner combined with the reflow algorithm constitutes the global phase of placement.

Algorithm **Partitioner+Reflow** describes the partitioner and the reflow algorithm applied during a placement cut. In general, each of these steps could be applied independently. For many chips, the combination of partitioning followed by reflow seems to work well. At any point before or after partitioning and reflow, other synthesis and placement transforms

Algorithm DetailedPlaceOpt

```
/* Detailed Placement Optimizations */
/* Create a small window, approximately large enough for 20 objects */
slider_window = initialize_it;
/* Slide this window across the entire chip */
for(postition = first; position != NULL; position = next)
      position_slider_window(slider_window,position);
      for_each_object_in_slider_window
         try swapping with each of the other objects in the window
         pick several objects, and try all permutations of reordering
         accept the best of all of these moves;
         legalize move
      end for;
end for;
```

may be applied. In one typical TPS scenario described in Section 5, synthesis transforms are applied immediately after reflow.

The detailed placement transforms involve the use of a collection of greedy heuristics that involve selecting an object for a move, or a pair of objects for a swap. A detailed placement transform of *swap* is given in algorithm **DetailedPlaceOpt**. Following the move or swap, a legalizer cleans up any overlaps. Next, we score the result. The scoring function includes timing, noise and area objectives. If we have improved the design we keep the move and bad moves are rejected.

4.2 Circuit Migration

The circuit migration transform allows one to have more precise control over the placement of cells to meet specific targets. The main difference between placement algorithms such as reflow and the algorithm presented in this section is the direct tight coupling with the analysis tool, in this case the timing analyzer.

In order to improve the timing of a design by moving appropriate circuits, we need to understand how circuit motions affect the capacitance of a net or a set of nets. Many situations occur where individual circuit movements have no effect or could even worsen the timing of the circuit. On the other hand the collective motion of several circuits together may have significant improvements. In the following examples we will make the simplifying assumption that timing is directly proportional to the net length.

Consider the meander in a critical path in Figure 3. Moving only one of the circuits C, D or E would have no beneficial effect on the total net length. The movement of C, D and E together would reduce total net length and therefore improve timing. The problem becomes more complex if C, D or E have multiple fanins and fanouts. The circuit migration transform addresses this issue. Similarly, consider a single net with three nodes connected using a steiner tree as shown in Figure 4. If we move any individual node A or B in the vertical direction, there is no reduction in net length (assuming orthogonal routing). If we move the two nodes together as shown the total net length can be decreased.

A circuit migration transform to perform timing optimization in conjunction with an incremental timing analyzer based on the notion of *strong moves* has been developed in [8]. A *strong move* involves moving an optimal set of circuits to improve the timing and has the property that moving any proper subset of this set will result in a suboptimal or no improvement. For example in Figure 3 the motion of C, D, E together in the downward direction is a strong move. A strong move for a net consists of moving a set of circuits all of which are connected to that net. Likewise, a strong move for a group of nets consists of moving a set of circuits all of which are connected to at least one net in the group.

Circuit migration transform uses efficient techniques [8] to compute the set of strong moves which improve timing for all individual nets in the critical region identified by the timing analyzer. Next, it combines these strong moves to generate strong moves for a group of nets in the critical region. Such strong moves are applied if the placement bin capacities are not exceeded and if there is significant timing improvement.

Algorithm LogicalEffortNetWeight

```
initialize_min_cut_placement()
logical_efforts = analyze_library()
number_of_cuts = 0
while(cuts to be processed)
      CR = obtain_critical_region(design);
      if (mode = absolute)
         for each net in CR
            absolute_slack_weight = compute_slack_weight(net)
            net_weight = f(absolute_slack_weight,logical_effort/max_logical_effort)
         end for;
      else if (mode = incremental)
         for each net in CR
            absolute_slack_weight = compute_slack_weight(net)
            previous_slack_weight = previous_slack_weight(net)
            new_slack_weight = φ(absolute_slack_weight,previous_slack_weight)
            net_weight = f(new_slack_weight,logical_effort/max_logical_effort)
         end for;
      end if;
end while;
```

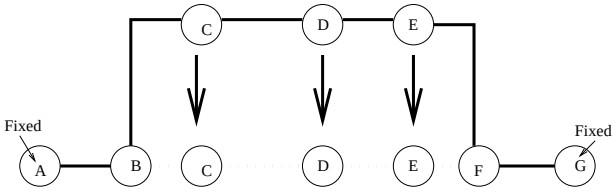

Figure 3. Changes in Locations for Critical Paths

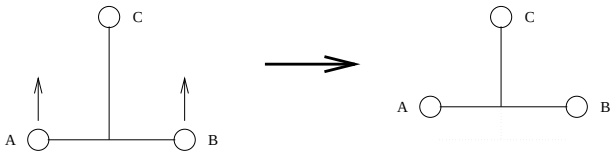

Figure 4. Motion of individual Circuits

4.3 Logical Effort based Net Weights

Timing driven placement [9, 20, 1] techniques often used in typical chip design methodologies [15] incorporate constraints (such as net weights, capacitance/delay budgeting

etc.) into a placement algorithm to improve the locations for timing. One problem with selecting net weights prior to placement is that the weights are determined by the timing sign-off obtained after synthesis. Synthesis typically operates on wire load models, and may predict the critical paths incorrectly. Assigning net weights and budgeting delay would bias the placement to optimize paths that may not be critical after placement. Further, net weights often do not take into account the delay sensitivity of a given gate type to the capacitance resulting from the wire length.

The techniques presented in this section, allow the use of net weights to control the timing more precisely. Net weights are updated on the nets during each cut as the placement gets more refined. The net weights are not only scaled according to how timing critical the nets are, but are also scaled according to the logical effort [24] of the gates driving the nets.

In the gain based synthesis technique used [24, 3], the delay of a gate is determined by its gain and is independent of the load driven by the gate as given in equation 1.

$$d = k_0 + k_1 * gain \quad (1)$$

where $gain = C_l/C_{in}$, C_l is the capacitative load the output pin drives and C_{in} is the input pin capacitance of the corresponding input pin of the gate (each arc from input to output has an equation of the form 1).

The *logical effort* of a gate type (say nand, nor, invert) gives a measure of the delay sensitivity of a gate type to gain. Logical effort can be used to decide the capacitances that gates in a netlist should drive relative to each other. For example gates with lower logical efforts (such as static inverters) are preferred to drive larger capacitances and wires whereas higher logical efforts (such as static xors) are used only to drive smaller loads. Scaling weights in proportion to the logical effort is similar to designers rule of thumb where more complex gates are allowed to drive only shorter wires while simpler gates such as inverters and nand gates are preferred to drive longer wires. The concept of logical effort allows us to automatically incorporate these rules of thumb.

In algorithm **LogicalEffortNetWeight**, the library is analyzed prior to placement to obtain the logical efforts of the gates in the library. On each cut of the placement, a new critical region is obtained and the weights are estimated for the nets. In the *absolute* mode new weights are calculated and assigned on each cut independent of any previous weights. If the *incremental* mode is chosen, the previous net weights are used in the computation of the new net weights, allowing a smoother change in the assignment of net weights.

4.4 Gate Sizing

Gate sizing is traditionally performed to both optimize the performance of critical paths and to satisfy the electrical constraints imposed by the design rules. In addition, sizing also performs area recovery on the non-critical regions. A large variety of gate sizing algorithms are available ranging from non-linear optimization problems with sophisticated, run-time intensive optimizations to quick and fast heuristics. In TPS the accuracy of the applied gate sizing algorithm increases as the placement transforms progress. This means that the precision of the gate sizing is controlled and more precise optimizations are applied as the physical locations of gates and therefore wire load estimates become more accurate.

Algorithm PlacementDisc

```
initialize_min_cut_placement()
initialize_wire_estimates()
initialize_steiner_based_timing_analysis()
cut_status = 0
while(cuts to be processed)
    if (cut_status == T)
       Execute Discretization, link cells
       Update timing analysis mode to actual
    end if;
    if (cut_status < T)
       Timing analysis mode is gain based
       Execute virtual discretization
       (sized cells not linked)pass on netlist
       Provide width and height of cells to placement
    end if;
    if (cut_status > T)
       Timing analysis mode is based on actual sizes
       Perform other timing optimizations
    end if;
    update_wire_estimates()
    update_steiner_based_timing_analysis()
end while;
```

The concept of gain introduced in section 4.3 is used in the process of gate sizing. Prior to placement, the gates are modeled as sizeless cells (only a gain value is assigned to each gate). During placement, a process of discretization is performed. The size of each gate is derived from the gain and the load (the sum of wire load and pin loads). An appropriate match from the library for the size is then obtained.

As shown in Algorithm **PlacementDisc**, *virtual* discretization is applied until a threshold cut status T is reached. This means that although the algorithm for discretization is applied on each instance in the design, the solution cell from the library is found but is not instantiated in the design. That is, by applying discretization, the size and shape of the physical cell which matches the given load are provided to the placer, but the timing analysis is not updated to reflect this choice. Virtual discretization does not cause the incremental timing analysis [10] to recompute. This makes a major difference to timing analysis, since performing *actual* discretization would result in re-implementation of the timing graph and therefore can be expensive in terms of run time.

After discretization, the sizes of the gates can be further tuned as the cuts progress. In most libraries, different drive-strengths with the same foot-prints are available. In TPS one can compensate for the difference between the estimated wire length and wire lengths resulting from actual routing by performing a final in-footprint gate sizing after routing.

4.5 Clock Tree and Scan Chain Net Length Optimization

Clock tree optimization aims to minimize the total length of nets in a clock tree between registers and clock buffers so as

to maximize performance and minimize clock skew. In most designs the amount of registers and number and size of clock buffers require that clock tree optimization be an integral part of TPS.

Traditionally, an initial placement is done ignoring clock nets so that registers are assigned locations based on their data connections [15]. Next, clock tree optimization is performed by assigning locations and connections to clock buffers based on register locations.

The fact that clock blocks are typically much larger than registers (and other gates) creates a major disturbance at this relatively late stage in placement. It is very hard to create enough legal space around the new location of the clock blocks. In addition, holes created by moving these large clock buffers from their original locations may not be very well utilized. Fixing the topology of the clock trees before hand takes away the flexibility to optimize this and let the data-flow dominate the register placement.

TPS addresses the problem as follows. Initially, net weights on clock nets are set to zero so that they are ignored by the placement transforms. In addition, the sizes of clock buffers are reduced to zero and the sizes of the corresponding registers are increased to account (or save space) for clock buffers. After placement has progressed sufficiently (say about 30 % of the final placement), net weights on clock nets are restored to their original values. The sizes of clock buffers and registers are also restored to their original sizes. Note that restoring registers to their original sizes results in free space in the bins containing the registers. Next, clock net optimization is performed which takes advantage of this free space in assigning locations to clock buffers. As a result, typically very little or no overlap is created by clock net optimization. Placement then progresses and utilizes any free space left over after clock buffer location assignment.

Scan chain optimization aims to minimize the total length of nets in a scan chain. This is typically done by reordering registers in scan chains once locations of registers are known. Similar to clock net optimization, net weights on pure scan nets (scan nets with no data connections) are initialized to zero so that they are ignored by placement transforms. After placement is nearly complete (say about 80 % of final placement), net weights on scan nets are restored to their original values. Next, scan chain optimization is performed by reordering registers in scan chains based on the locations of registers to minimize total length of scan nets.

Algorithm **Clock and Scan Net Optimization** summarizes the above description of clock and scan net length optimization. Please refer to Section 5 for more details about *cut_status* variable.

Algorithm Clock and Scan Net Optimization

```
initialize_min_cut_placement()
cut_status = 0
while(cuts to be processed)
    if (cut_status == 10)
      initialize all net weights to default value
      set clock net weights to 0
      set scan net weights to 0
      reduce clock buffers sizes to 0
      increase register sizes
    end if;
    if (cut_status == 30)
      set clock net weights to default value
      restore clock buffer sizes
      restore register sizes
      perform clock net optimization
    end if;
    if (cut_status == 80)
      set scan net weights to default value
      perform scan net optimization
    end if;
end while;
```

4.6 Circuit Relocation

In addition to the transforms described earlier, all existing synthesis optimizations such as cloning, buffer insertion, remapping and so on have been adapted to work with the TPS environment. These algorithms have been modified in such a way that the netlist changes they cause will result in minimal perturbation to the existing placement.

Consider a transform that attempts to clone gates in order to improve timing. During the evaluation phase, it may determine that the clone has to reside in the same bin as the cloned gate. If the current space in the bin is not sufficient, it is necessary to create space within a bin without adversely affecting the worst case timing.

In most cases it is possible to move noncritical cells away from the local bin to allow timing optimization on critical cells. A mincost network optimization algorithm, called circuit relocation has been developed which determines the best combination of bin to bin cell moves that frees the local area for timing optimizations. Circuit relocation is a placement utility that is either called as a stand-alone transform or from within a single transform to explicitly create space in a certain bin.

5 TPS Scenario

The flexibility of the transformational approach allows us to easily develop specific scenarios tuned to take advantage of a converging design process. Accuracy versus runtime tradeoff for various transformations can be selected as the quality of the placement and netlist data improves in a converging flow. Such scenarios can be developed to target a variety of metrics including noise, yield and manufacturability. The scenario presented in this section specifically targets timing optimization while maintaining the wirability metrics.

In our system, technology independent optimization, technology mapping and the early part of the timing optimization stage (where we do coarse optimization) employ a gain-based (load-independent) delay model [3]. As a result, the effect of wire load models on area-delay tradeoffs performed is minimized. The later part of timing optimization, where detailed and aggressive optimization is performed, is integrated with transformational placement. Global optimization transforms are employed in the initial stages of placement. On the other

hand, local and detailed optimization transforms which tend to cause very minor perturbation to the placement, are used in the later stages of placement. Placement transforms such as Partitioner, Reflow, and Detailed Placement are invoked periodically to bring the design into a desired state so that the other transforms such as those described in Section 4 are applied.

At any point during the process, the progress of placement is measured by the size of the placement bins. The Partitioner transform is invoked to convert an existing placement to one with bins of desired size. Partitioner provides the status or progress of the placement by providing a number between 0 and 100 based on the bin sizes. Low numbers imply initial stages of placement while higher numbers are returned for later stages. At any time, Partitioner may be invoked with a target status number greater than the existing status number. Partitioner will then proceed with placement and attempt to bring the design into a state with status number as close as possible to the target status number. The Reflow transform is typically invoked after Partitioner to improve the placement. In our approach, we let the placement advance in steps of a specified number and selectively apply transforms once the desired state of placement is reached. The step size may be user specified or derived from the design size and other properties.

TPS_scenario in Figure 5 gives a high level description of the optimization process. First, the timing analyzer, wire length calculator, and clock tree optimizer are initialized. The placement *status* range at the beginning of some blocks gives the condition under which that block is executed. For example, Circuit Migration transform is applied only if *status* is between 30 and 50. On the other hand, Clock Optimization is performed only once when *status* is 30. During the initial stages of placement, gate sizing is performed in the non-critical regions of the design to recover area, as more realistic wire loads are available. The clone transform which makes cloned copies of gates to distribute load, and the buffer insertion transforms are applied during the middle stages of placement. Note that clone and buffer insertion transforms take care to avoid overlap and congestion while assigning locations to the newly created gates. These transforms also utilize the circuit relocation transform to create space for the newly created gates. The output of this system is a fully synthesized and legally placed design that can be input to the routing tool. Post-routing a final in-foot-print gate sizing (which does not disturb placement or routing) is done to compensate for mismatches in actual and Steiner tree predicted routing.

6 Results

The results of experiments are given in Table 1. The experiment was to compare the TPS scenario described in the last section with traditional iterative loop of separate synthesis and placement steps (SPR). The goal was to optimize timing while maintaining other placement metrics such as wirability.

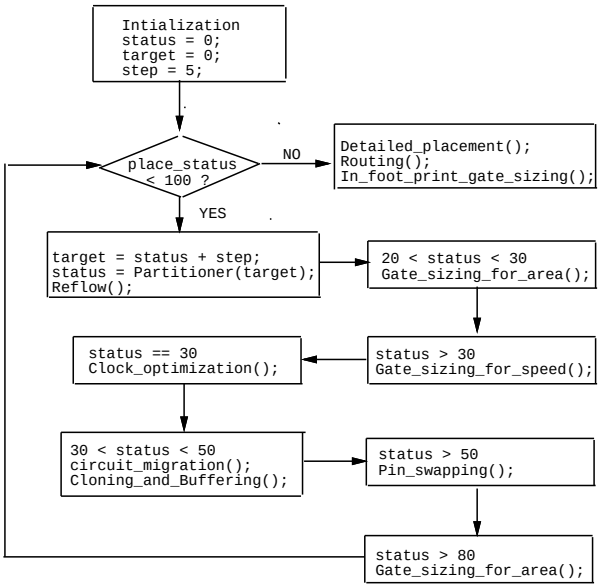

Figure 5. Optimization Flow Chart

Table 1. Results for TPS

Ckt	Flow	Area	slack	% cycle time	Horiz	Vert
		icells		impr.	pk/avg	pk/avg
Des1	SPR	18622	-380		295/224	305/234
	TPS	16510	-222	6.5	273/207	292/227
Des2	SPR	25927	-376		376/275	307/248
	TPS	23742	-168	8.6	426/305	320/273
Des3	SPR	39734	-364		461/324	584/453
	TPS	37136	-192	7.1	472/357	746/642
Des4	SPR	21584	-410		343/248	277/202
	TPS	19736	-134	11.5	403/290	333/249
Des5	SPR	14780	-230		226/180	270/216
	TPS	12390	-56	7.25	203/151	272/205

We used 5 partitions of a mainframe processor. Synthesis-placement-resynthesis(SPR) is compared to the TPS. A proprietary synthesizer and commercial quadratic placer were used in SPR. In all test cases we have a timing improvement (upto 11%) of the cycle time. In all testcases, the area of both the SPR and TPS runs were about the same with a slight improvement in the case of TPS. More importantly these results are obtained with a single invocation of TPS compared to iterations till timing closure was acheived for SPR. The timing improvement is significant since the designs were highly optimized through many iterations of placement and synthesis in SPR to meet aggressive timing constraints.

Wirability was measured in terms of the horizontal and vertical wires cut, and both the peak and the average wires cut are given. The wireability did not increase significantly and we could route all chip partitions after TPS. There is a slight increase in congestion since the partitions were primarily tuned for timing.

The CPU times for SPR included repeated steps of synthesis and placement as well as manual intervention. The CPU time for TPS on the other hand was equal to about *one* run

of synthesis followed by placement. Therefore TPS run times were significantly better.

7 Conclusions

A transformational approach where placement and synthesis transforms efficiently and concurrently manipulate the design space is presented.

Our ability to apply fine grained transformations at varying levels of accuracy allows placement and synthesis to progress into a single converging flow. We therefore avoid costly and unpredictable standalone placement and synthesis iterations, used in traditional design methdologies.

The results indicate a significant improvement in the timing of previously optimized designs while maintaining traditional measures of placement like total wirelength and wirability.

Recent work has involved extending the methodology and algorithms to handle full chips of about a million gates flat. The preliminary results on these chip sizes are consistent with those presented in the previous section with reasonable run times of a few hours. Other work involves extending algorithms to optimize metrics such as noise, congestion, power and yield.

References

[1] C. Alpert, T. Chan, A.B.Kahng, I. Markov, and P. Mulet. Faster minimization of linear wirelength for global placement. *IEEE Transactions on Computer-Aided Design*, 17(1), Jan. 1998.

[2] C. Alpert, D.Huang, and A.B.Kahng. Multilevel circuit partitioning. In *Proc. ACM/IEEE Design Automation Conference*, pages 530–533, 1997.

[3] F. Beeftink, P. Kudva, D. Kung, and L. Stok. Gate size selection for standard cell libraries. *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1998.

[4] B.Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computer-Aided Design*, pages 438–446, 1984.

[5] J. Cong, L. He, C.-K. Koh, and P. H. Madden. Performance optimization of VLSI interconnect layout. *Integration*, 21:1–94, 1996.

[6] W. E. Donath. Equivalence of memory to random logic. *IBM Journal of Research and Development*, pages 401–407, September 1974.

[7] W. E. Donath. Wire length distribution for placements of computer logic. *IBM Journal of Research and Development*, pages 152–155, May 1981.

[8] W. E. Donath, P. Kudva, and L. Reddy. Performance optimization of network length in physical placement. In *Proc. International Conf. Computer Design (ICCD)*, pages 258–265, Oct. 1999.

[9] W. E. Donath, R. J. Norman, B. K. Agrawal, S. Y. H. S. E. Bello, J. M. Kurtzberg, P. Lowy, and R. I. McMillan. Timing driven placement using complete path delays. In *Proc. ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, June 1990.

[10] D. Hathaway, R. Abato, A. Drumm, and L. van Ginneken. Incremental timing analysis. Technical report, 1996. IBM, U.S. patent 5,508,937.

[11] S. Hojat and P. Villarrubia. An integrated placement and synthesis approach for timing closure of PowerPC microprocessors. *Proc. International Conf. Computer Design (ICCD)*, pages 206–210, 1997.

[12] L. Kannan, P. R. Suaris, and H.-G. Fang. A methodology and algorithms for post-placement delay optimization. In *Proc. ACM/IEEE Design Automation Conference*. IEEE Press, June 1994.

[13] G. Karypis and A. Kumar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proc. ACM/IEEE Design Automation Conference*, pages 526–529, 1997.

[14] K. Kleinhans, G. Sigl, F. Johannes, and K.J.Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, pages 356–365, 1991.

[15] K.L.Shepard, S.M.Carey, E.K.Cho, B.W.Curran, R.F.Hatch, D.E.Hoffman, S.A.McCabe, G.A.Northrop, and R.Seigler. Design methodology for the S/390 parallel enterprise server g4 microprocessors. In *IBM Journal of Research and Development*, pages 515–547, July/September 1997.

[16] M. T.-C. Lee and et. al. Incremental timing optimization for physical design by interacting logic restructuring and layout. *International Workshop in Logic Synthesis*, pages 508–513, 1998.

[17] J. Lou, A. Salek, and M. Pedram. Exact solution to simultaneous technology mapping and linear placement problem. *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1997.

[18] M. Murofushi, T. Ishioka, M. Murakata, and T. Mituhashi. Layout driven re-synthesis for low power consumption lsis. In *Proc. ACM/IEEE Design Automation Conference*. IEEE Press, June 1997.

[19] L. Pileggi. Timing metrics for physical design of deep submicron technologies. In *Proc. International Symposium on Physical Design*, pages 28–33, 1998.

[20] M. Sarrafzadeh, D. Knol, and G. Tellez. Unification of budgeting and placement. *Proc. ACM/IEEE Design Automation Conference*, pages 758–761, 1997.

[21] A. Srinivasan, K. Chaudhary, and E.S.Kuh. RITUAL: A performance driven placement algorithm for small cell ICs. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 48–51, Nov. 1991.

[22] G. Stenz and et.al. Timing driven placement in interaction with netlist transformations. *Proc. International Symposium on Physical Design*, 1997.

[23] L. Stok, D.S.Kung, D.Brand, A.D.Drumm, A.J.Sullivan, L.N.Reddy, N.Hieter, D.J.Geiger, H.H.Chao, and P.J.Osler. Booledozer:logic synthesis for ASICS. *IBM Journal of Research and Development*, July 1996.

[24] I. E. Sutherland and R.F.Sproull. Theory of logical effort: Designing for speed on the back of an envelope. In *Advanced Research in VLSI: Proceedings of the 1991 University of California Santa Cruz Conference, C. Sequin Ed. The MIT Press*, 1991.

[25] W.C.Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(1):55–63, 1948.