Submitted to Professor Marek-Sadowska on December 12, 1996, as a requirement for Introduction to Design Automation, ECE 256A.

Incremental Timing Driven Placement

Steve Haynal
e-mail: haynal@engineering.ucsb.edu

Abstract

Standard cell layouts may need only slight modifications to meet timing constraints. In these situations, general purpose algorithms, which consider numerous parameters of the layout, may be too time consuming or too coarse to make the changes where needed. This paper presents an incremental timing driven placement algorithm designed to "cleanup" a handful of critical paths in a previously placed standard cell layout. It does this by minimizing *RC* delay contributions from wiring between critical nodes, and wiring to and gate sizes of periphery nodes on critical nets. Modifications are made in a way that minimally perturbs the existing layout.

1. Introduction

Creating a standard cell layout requires selecting appropriate standard cells from a library and arranging them cost effectively in two-dimensional finite space. A completed and interconnected placement, requiring the smallest possible area yet still satisfying sets of design, technological, and performance constraints is considered most cost effective. Because of the layout problem's difficulty, heuristic methods are used to generate good layouts which are only approximations of the optimum layouts. This implies that there is room for improvement in a standard cell layout.

The best way to take advantage of this room for improvement may not be by changing the heuristic used when completing a layout from scratch. These heuristics usually try very hard to incorporate algorithms that are linear in complexity. By doing so, they bypass methods that might produce better results for the sake of time and computational complexity. On the other hand, incremental algorithms can afford to use more complex heuristics since they only consider a small set of critical cells. More optimum and timely results may be achievable by first completing a layout from scratch using a linearly complex algorithm and then fine tuning it using a more complex algorithm which only focuses on problem areas.

This paper presents an incremental timing driven placement algorithm designed to "cleanup" a handful of critical paths in a previously placed standard cell layout. It does this by minimizing *RC* delay contributions from wiring between critical nodes, and wiring to and gate sizes of periphery nodes on critical nets. Modifications are made in a way that minimally perturbs the existing layout. Section 2 describes the underlying principles behind this algorithm. Section 3 presents the entire algorithm. Section 4 contains results. Finally, section 5 discusses observations and conclusions concerning this algorithm.

2. Underlying Principles

Prime contributors of delay in VLSI standard cell designs are *RC* effects of wiring and gates sizes. Figure 1 shows possible delay contributions along a critical path. Nodes A and B are along the critical path that enters from the left and leaves to the right. The net between nodes A and B is a critical net. Nodes E and C contribute to the overall critical net *RC* delay because of their larger

gate sizes and consequently larger input capacitance. Nodes E and D also contribute to the *RC* delay because connecting wire lengths are not minimized. Obviously, the *RC* delay of this critical net can be reduced if E and C's gate sizes are minimized and E and D are swapped with cells X and Y respectively.

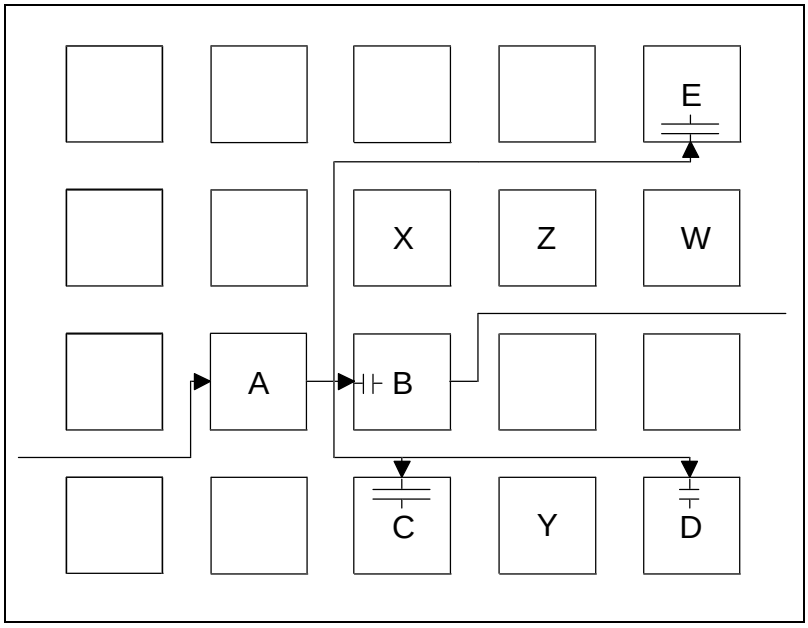

Figure 1: Delay contributions to one net in a critical path.

If these simple principles are applied to all nets in a critical path, the result would be a cluster of cells around some fixed point or strings of cells between a number of fixed points. This is not an ideal solution because there is a balance that must be maintained. If one critical path is optimized by clustering cells, then another previously non-critical path may become critical. Therefore, any incremental solution should make changes in a way to minimize global impact. For example, instead of directly swapping E and X, perhaps it would be less perturbing to the system to first swap E and W, then E and Z and finally E and X. Finally, these principles need not be applied until clustering is achieved. Stopping once a critical path delay has been reduced sufficiently will minimize perturbation to the surrounding cells.

This algorithm has six underlying principles. First, as described above, critical cell clusters represent the ideal solution for one critical path at the expense of other paths. Therefore, this

algorithm will try to cluster critical cells, first for a critical net and then for a critical path, if allowed to execute indefinitely. Second, since large gate sizes increase *RC* delay, all gates connected to critical nets will be reduced as much as allowable. [2] Third, as prompted by [1], only localized movement will be considered. This will spread out impact costs and minimize the chance of other previously non critical nets and cells becoming critical. Fourth, a probabilistic hill climbing step will be included to reduce the chance of getting stuck in a local minimum. Reasoning used by Kernighan and Lin in [4] will provide the necessary hill climbing attribute. Fifth, good path delay estimates are needed to identify critical paths and evaluate modified placements. The Penfield and Horowitz model will provide this. [3] Finally, this algorithm will stop when its goal is reached. This protects against drastic secondary problem producing changes that might occur if the algorithm runs to total optimization for the selected critical paths. Also, this allows for greater freedom to concentrate on more problematic paths once less problematic paths have been solved.

3. An Incremental Timing Driven Placement Algorithm

This algorithm has seven main steps.

1. Input of standard cell placement, connectivity, buffer sizes, critical and near critical paths and desired critical path delay reduction.
2. Identification of critical cells, critical nets and near critical nets.
3. Calculation of net centers, ideal cell locations, critical net goal costs and global cost.
4. Greedy localized movement of critical cells towards their ideal locations.
5. Kernighan-Lin style back tracking to the lowest cost point in the sequence of moves.
6. Iterations and completion.
7. Down sizing off all critical cells and timing driven buffer sizing.

Step 1: Input of standard cell placement, connectivity, buffer sizes, critical and near critical paths and desired critical path delay reduction.

Since this algorithm is incremental, it assumes that a fairly finalized layout exists. Placement, connectivity and buffer size data is therefore readily available and provided to this algorithm. The placement and connectivity input also identify movable standard cells and fixed pin locations. Furthermore, since the motivations to use this algorithm are known critical paths, timing information is also available. Critical and near critical paths, including categorization of delays into ballistic and wiring, are provided to this algorithm.

Step 2: Identification of critical cells, critical nets and near critical nets

Connectivity and critical path data from step 1 is used to form a critical paths list. Each critical path in this list contains a critical nets list and critical cells list. The critical nets list contains all nets connecting nodes in the critical path. Critical nets are given a weight of 2. Near critical nets are identified in the same manner but receive a weight of 1. All other nets in the layout receive a weight of 0. Finally, a critical path's critical cells list contains **all** cells (not just critical nodes) connected to its critical nets.

Step 3: Initial calculation of net mass centers, ideal cell locations, goal costs and global cost.

A center of mass for all critical and near critical nets is calculated using the following formulas:

$$x_{net} = \frac{1}{n} \sum_{i=1}^{n} x_{cell_i}$$

where x_{cell} is the center x coordinate of a cell connected to that critical net and n is the total number of cells connected to that critical net. Likewise,

$$y_{net} = \frac{1}{n} \sum_{i=1}^{n} y_{cell_i}.$$

An ideal cell location is calculated for each critical cell using the following formulas:

$$x_{ideal} = \frac{\sum_{j=1}^{m} \left(w_j \times x_{net_j} \right)}{\sum_{j=1}^{m} w_j}$$

where w is the net weight and m is the number of nets connected to this particular cell. Likewise,

$$y_{ideal} = \frac{\sum_{j=1}^{m} \left(w_j \times y_{net_j} \right)}{\sum_{j=1}^{m} w_j}.$$

A cell's ideal location provides a movement direction vector to minimize critical nets. This ideal location changes as the algorithm proceeds.

Step 3 also calculates a goal cost for each critical path. The cost for a critical path is

$$pathc = \sum_{j=1}^{n} \left(w_j \times \beta_j \right)$$

where β is the bounding box half perimeter of a net and n is the total number of critical nets along the critical path. Given the total delay due to wiring, d_{wiring}, and the desired time reduction for a critical path, d_{reduce},, the goal cost for a critical path is roughly estimated as

$$goalc = \frac{d_{wiring} - d_{reduce}}{d_{wiring}} \times pathc_{initial}.$$

Finally, step 3 calculates a global cost using

$$globalc = \sum_{j=1}^{m} \left(w_j \times \beta_j \right)$$

where m is the total number of unique critical and near critical nets on all critical paths.

Step 4: Greedy localized movement of critical cells towards their ideal locations.

To begin, the first critical path is picked from the critical paths list and its first critical cell is picked for movement. A move set that is most in the general direction of this cell's ideal location,

is picked. The eight possible moves sets are shown in Figure 2. Notice that the move sets contain only seven local and hopefully minimally perturbing moves.

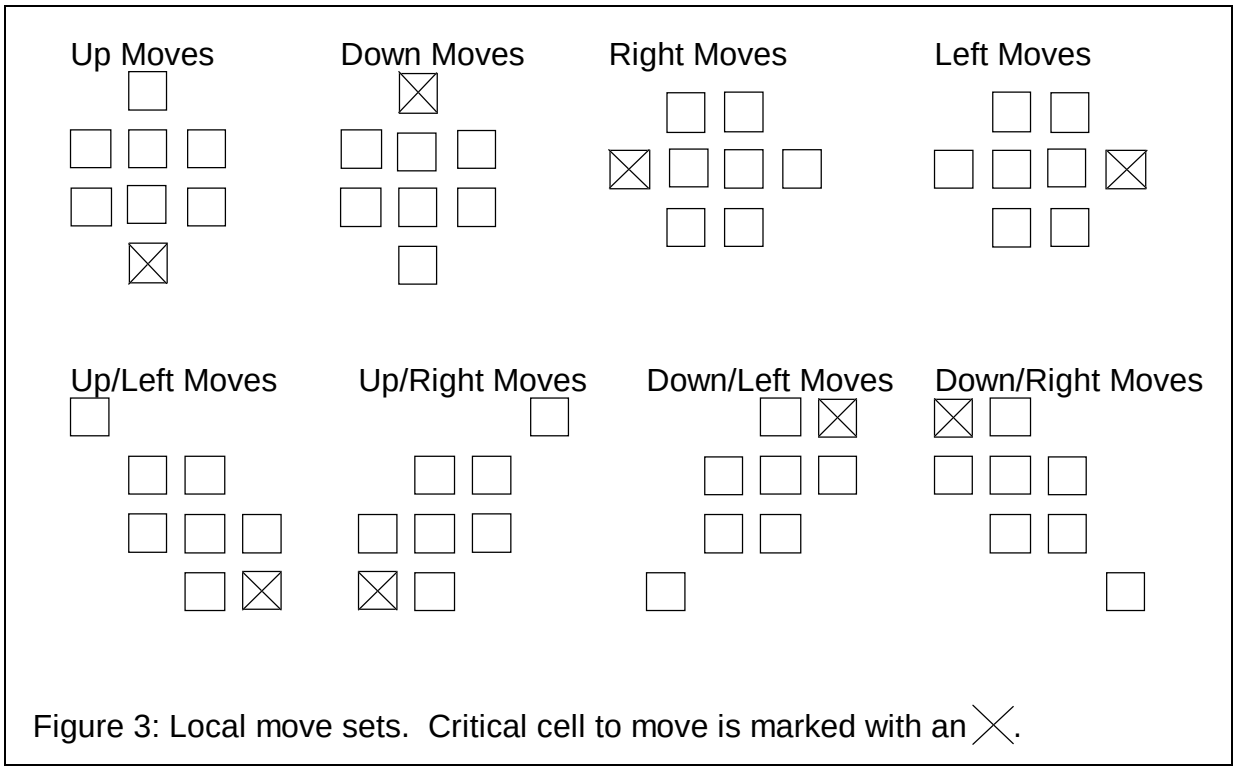

Figure 2: Possible critical cell move sets.

Since standard cells vary in size and there are dense and not so dense areas in standard cell layouts, the algorithm attempts to move the critical cell to one of these new locations according to the following sequence of trials. If any trial is successful, no further trial is attempted.

A. Jam trials
   1. If there is space on the preferred side of the target cell, jam the critical cell into that space. The preferred side is the side closest to the ideal location for the critical cell.
   2. Attempt a jam on the non-preferred side of the target cell.
   3. Shift the cell on the preferred side of the target cell as far away as possible and attempt a jam on the preferred side of the target cell.
   4. Shift the cell on the non-preferred side of the target cell as far away as possible and attempt a jam on the non-preferred side of the target cell.

B. Swap trials

1. Attempt to swap the target and critical cells.
2. Shift the cell on the preferred side of the target cell as far away as possible and attempt to swap the critical and target cells.
3. Shift the cell on the non-preferred side of the target cell as far away as possible and attempt to swap the critical and target cells.
4. Attempt to swap the critical cell with the target cell and its preferred neighbor.
5. Attempt to swap the critical cell with the target cell and both of its neighbors.

Each time one of the seven moves in the move set is tried, an accept cost estimate is made using the following formula:

$$accept = \sum_{i=1}^{n} \left(w_i \times \left(\beta_{old_i} - \beta_{new_i} \right) \right)$$

where n is the total number of critical and near critical nets effected by this move. The move providing the best accept value, even if it is negative, is greedily taken. This cell is marked as fixed and the accepted move is stored. Step 4 is repeated until every critical cell on the critical path's critical cells list has been moved once.

Exceptions for Step 4: Greedy localized movement of cells toward the critical path mass center.

If a critical cell in step 4 is already in its ideal location or can not move because of constraints (surrounding fixed cells, a large cell in a dense area, etc.) it will attempt a move towards the critical path center of mass. This critical path center of mass is calculated as:

$$x_{cp} = \frac{\sum_{j=1}^{m} \left(w_j \times x_{net_j} \right)}{\sum_{j=1}^{m} w_j}$$

where m is the number of nets in this critical path. y_{cp} is calculated similarly. This is done to provide some impetus for cells to eventually glom together for each critical path.

Step 5: Kernighan-Lin style back tracking to the lowest cost point in the sequence of moves.

If n moves were made in step 4, n-(k+1) of these moves are undone so that

$$\sum_{i=1}^{k} accept_i$$

is maximized and greater than or equal to zero. The new global cost is set to

$$g_{new} = g_{old} + \sum_{i=1}^{k} accept_i .$$

Step 6: Iterations and completion.

After step 5 accepts a certain number of moves, all cells are flagged as moveable. The critical cell list for the current critical path is randomly reordered so that moves will be attempted in a different order next time. Net centers and ideal cell locations for all critical nets and critical cells in this critical path are recalculated as in step 3. The next critical path in this list becomes the current critical path and steps 4 and 5 are repeated for this critical path if it hasn't achieved its goal cost.

Once all critical paths have been optimized once, the critical path list is randomly reordered and steps 4-6 are repeated unless one of two conditions is met. First, if the goal cost for all nets has been achieved, the algorithm will proceed to step 7. Second, if no change in global cost has been noted after five passes through the list of critical paths, the algorithm will proceed to step 7.

Step 7: Down sizing of all critical cells and timing driven buffer sizing.

Each critical cell is reduced to the smallest buffer size possible. Notice that this even effects critical cells that are not nodes of the critical path Next, the critical paths undergo timing driven buffer sizing. This algorithm relies on another tool to do this timing driven buffer sizing.

4. Implementing and Testing the Algorithm

The incremental timing driven placement algorithm was implemented with an approximately 2200 line C++ program. This program was written to interface with Cascade's Epoch, a commercial custom VLSI CAD package. The main Epoch tools used to test this algorithm were Tactic, which provided Penfield-Horowitz delay analysis and timing driven buffers sizing, and the router, which allowed layout results to be inspected and quantified. To test the algorithm's effectiveness, it was used to cleanup eight known critical paths in a HP26G 0.8 μm technology 512 cell standard cell layout. These paths remained even after employing Epoch's standard cell group optimization and timing driven buffer sizing. The target clock period for this custom compression-encryption chip was 12.5 μs and the worst path exceeded this limit by roughly 900 ps. All eight errors are shown in Figure 3. Figure 4 shows the critical nets and critical cells for these critical paths in the original layout.

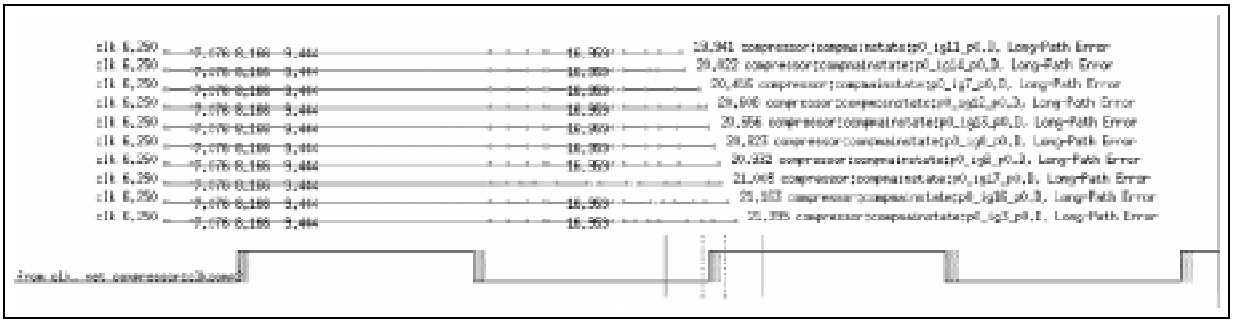

Figure 3: Initial critical paths in compression control logic.

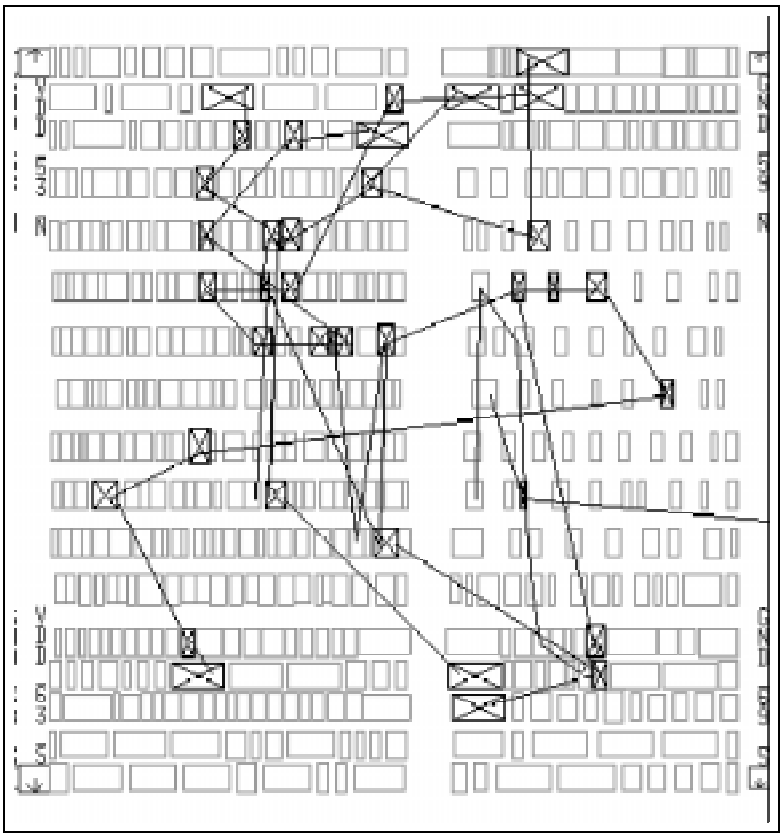

Figure 4: Initial critical net and critical cell placement

For the first trial of the incremental timing driven placement algorithm, the eight critical paths shown in Figure 4 were input as critical and the two paths at the top of Figure 3 were input as near critical. One fixed point, the net going out of the standard cell group in Figure 4, was input. The incremental timing driven placement algorithm was executed with d_{reduce} set to zero for all the critical nets. This forces the algorithm to glom critical cells for each path as much as possible. Figure 5 shows the results of this trial.

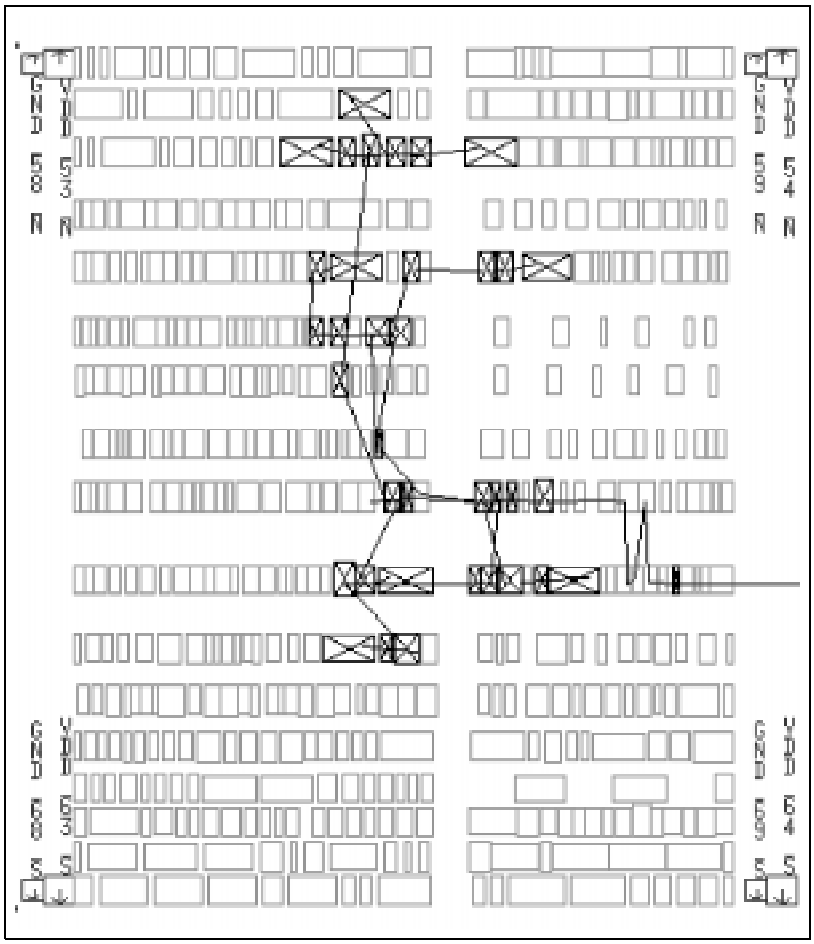

Figure 5: Maximum achieved glomming of critical cells.

Notice that substantial glomming of the cells have been achieved. This resulted after 20,285 moves and 28.42 seconds user time on a Sun Ultra Sparc 1. Every stored move and undone move is counted. This means that moves originally made and later undone in step 5 would count as 2 moves. The resulting design was larger, 2.06 μm^2 compared to 1.94 um^2 for the original layout. Only the worst initial critical path remained after being shortened by roughly 800 ps. This could have been eliminated if step 8 of the algorithm was executed.

After entering appropriate d_{reduce} times for each critical net, the algorithm was rerun with the same original layout data. A slightly better layout resulted after only 9905 moves and 14.49 seconds of user time. The resulting design size, 2.03 μm^2, was smaller than the first trial due to reduced cell displacement but still larger than the original. The improvement in the worst initial critical path was 850 ps; slightly better than the first trial. After executing step 8 of the algorithm, a final

layout, Figure 6, that met all the specifications resulted. The resulting layout was 2.02 μm^2 and the worst critical path was reduced by 1 ns, 100 ps under target. Figure 7 shows the final timing diagram for the modified layout. Eleven paths, one more than original, now lie in near critical zone. Notice that the eleven paths are more balanced than the original.

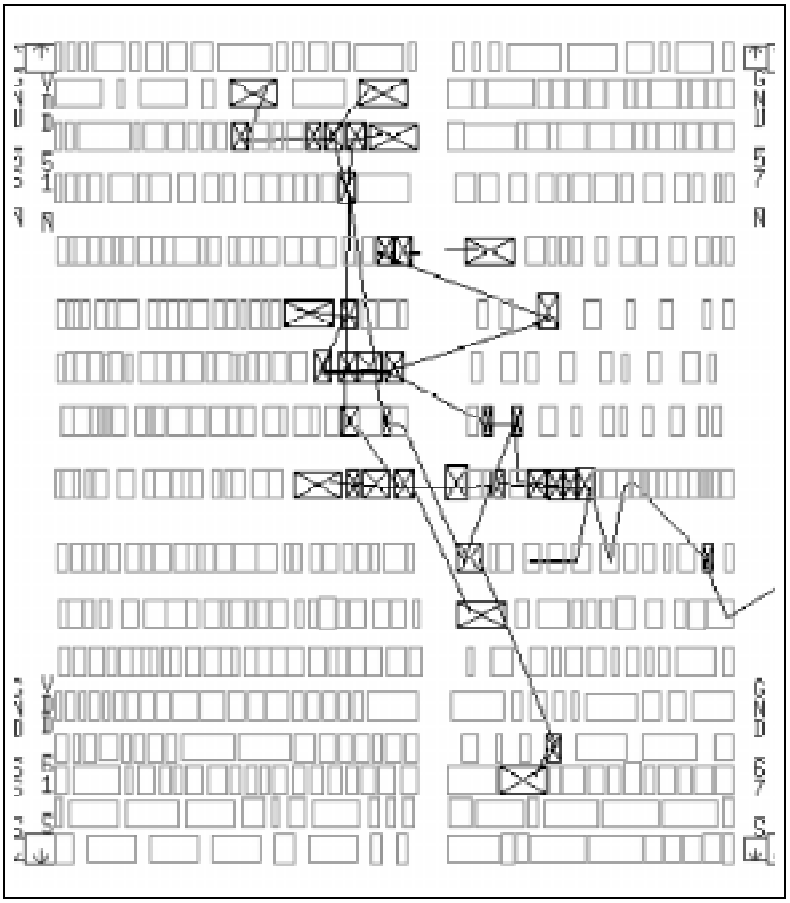

Figure 6: Final layout of compressor control logic

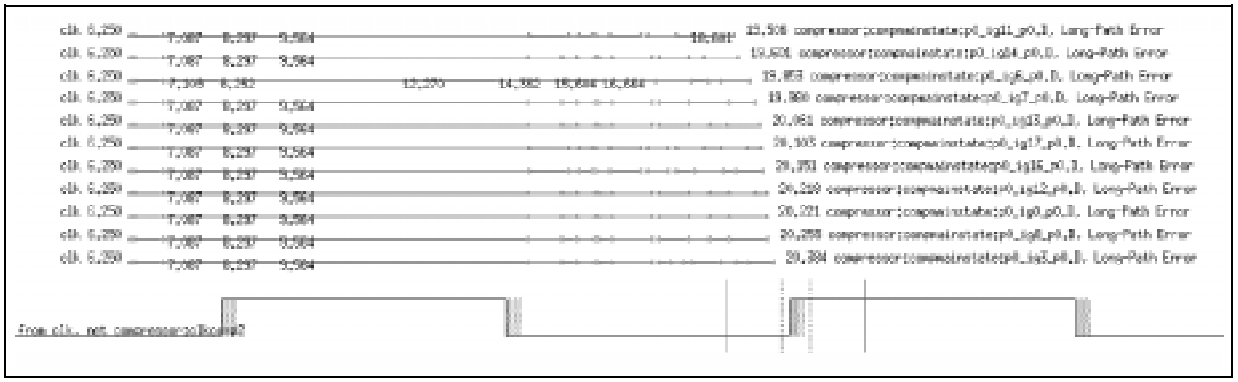

Figure 7: Final timing diagram of compressor control logic

5. Observations and Conclusion

This algorithm and its implementation could be improved in two ways. First, the implementation relied on a commercial CAD tool to do most of the timing analysis. A much better result should be obtained by incorporating timing analysis into the main loop of the algorithm. Second, the move choice was constrained to a maximum three-for-one swap. A *k*-for-one swap would result in more move freedom. Furthermore, the effectiveness of this algorithm could be better displayed if verified on standard cell layout benchmarks. Finally, a comparison to standard timing driven placement techniques would give a better perspective of this algorithm's power.

A new incremental timing driven placement algorithm is presented. Given a semifinal standard cell layout and timing data, this algorithm makes intelligent choices for local cell movement to achieve goal costs for a handful of critical path. The main iterative part of this algorithm is based on ideas similar to those used by Kernighan-Lin for partitioning and consequently the algorithm is approximately $O(n^3)$ where n is the number of critical cells. Since n is anticipated to be much less than the total number of cells, the time expense of a complete timing driven placement algorithm is avoided. Results obtained from a 512 standard cell test circuit showed that 11% clock speed increase could be obtained at a 4% area increase expense.

References

[1] C.S. Choy, T. S. Cheung and K. K. Wong, "Incremental Layout Placement Modification Algorithms," *IEEE Trans. Computer-Aided Design*, vol. 15, no 4, pp. 437-445, April 1996.

[2] W. Chuang and I. N. Hajj, "Delay and Area Optimization for Compact Placement by Gate Resizing and Relocation," *Proc. ACM/IEEE Design Automation Conf.*, pp. 145-148, 1994.

[3] J. Rubinstein, P. Penfield and M. Horowitz, "Signal Delay in *RC* Tree Networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, pp. 202-211, July 1983.

[4] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, 49, no. 2, pp, 291-307, 1970.