Timing Optimization of FPGA Placements by Logic Replication

Giancarlo Beraudo
ECE Department, University of Illinois at Chicago
851 S. Morgan St.,
Chicago IL, 60607
gberaudo@ece.uic.edu

John Lillis
CS Department, University of Illinois at Chicago
851 S. Morgan St.,
Chicago IL, 60607
jlillis@cs.uic.edu

ABSTRACT

Logic replication for placement level timing optimization is studied in the context of FPGAs. We make the observation that critical paths are dominated by interconnect delay and are frequently highly circuitous. We propose a systematic replication technique to "straighten" such paths. The resulting algorithm has several components: cell selection, slot selection for a duplicate cell, fanout partitioning and placement legalization. This algorithm is described and promising preliminary experimental results are reported with up to 29% improvement in critical path delay.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids

General Terms

Algorithms, Performance

Keywords

Timing Optimization, Logic Replication, Programmable Logic, Placement

1. INTRODUCTION

The idea behind logic replication is that by making copies of one or more logic cells, one can maintain the logical behavior of a netlist while, hopefully, enabling additional optimization. Consider a logic cell *a* and suppose that we have created a duplicate cell *a'*. The cell *a'* takes precisely the same inputs as *a* and produces exactly the same boolean function of those inputs as its output. In this situation, the pins in the circuit that need to receive this signal may now obtain it from either the output of *a* or *a'*. This added freedom has been exploited in numerous previous works in the literature.

Perhaps the most common way in which such freedom has been exploited in past work is for min-cut partitioning. Here, it is observed that by selectively replicating certain logic cells, or clusters of cells, the crossing count of a bipartition can often be significantly reduced, while of course maintaining logical equivalence. Representative work in this area includes [1], [2], [3] and [4]. Neumann et al. [12] integrate this idea into a timing-driven partitioning-based placement flow. Replication has also been exploited to enhance the performance of high fanout logic cells. It has been observed that by replicating such a cell so that, roughly speaking, one of the copies drives the non-critical sinks (and therefore can be comparatively small), and the other drives the critical sinks, overall performance and/or area improvements can be had. Thus, in such a formulation, finding a suitable partition of the sinks is the crucial task. Work in this area includes [6] and [7].

In this paper we exploit the ability to relocate the replicated cell so as to better optimize the paths flowing through it. There are two main premises motivating this idea. First, performance is increasingly determined not just by cell delays, but by interconnect delays; as a result meandering (non-monotone) signal paths from memory element to memory element degrade the clock period substantially. Figure 1 shows such an example. We say that such a path is non-monotone because the coordinates of the cells on the path do not follow a monotone order. Second, there is often a limit to the degree to which a timing driven placement tool can "straighten" such paths regardless of the sophistication of such a tool. Intuitively, this is because of structural properties of the netlist itself and the result is groups of cells that have multiple critical or near-critical paths flowing through them; the optimization of one such path (by moving said cell or cells) is at odds with the other paths. Replication with relocation gives a way to deal with this situation by allowing replicated cells to better serve various sub-paths without degrading others.

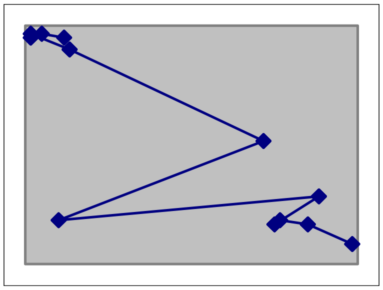

Figure 1: Example of non-monotone critical path (benchmark circuit Frisc placed with timing-driven VPR)

The work of Gosti et al. [10][11] has also used this idea at the level of Boolean network optimization coupled with placement. The present work differs in several ways. First, the stage of design at which the technique is applied is different in that we always operate on a mapped netlist rather than a Boolean network. Additionally, [11] has a different objective in that it attempts to optimize *all* input to output paths so that the block can perform well in any context when interfaced with other systems (this is an IP-based approach) while our focus is on the clock period of a single system and thus we consider only critical paths. The work differs in some technical areas as well including criteria and methodology for replication and legalization procedures.

For an initial case study of the potential of such techniques we focus on the FPGA domain for several reasons. From a pragmatic experimental point of view it is comparatively easy to construct a full FPGA design flow all the way to detailed routing for greater confidence in the effects of the optimization. This is enabled largely by the existence of the Versatile Place and Route system [5]. This allows us to easily track the effects on other design criteria such as wire-length. In practice, FPGA logic resources are also typically underused. This is because of the fixed architecture of FPGAs and as a result a typical placement will have many unused logic blocks. The existence of such blocks is a necessary enabler of logic replication. Finally, the delay characteristics of modern FPGAs is quite predictable due in part to the high degree of signal buffering and segmentation in most routing architectures. As a result, rectilinear distance is a reasonably effective first-order predictor of delay and this makes the notion of monotonicity an especially useful and clean abstraction for an initial study (as an aside, such an abstraction would seem particularly suitable for the standard cell domain for placement and replication optimization performed before buffer insertion which tends to linearize delay).

The contributions of the paper are summarized as follows. We first perform a preliminary analysis of the timing characteristics of placed FPGA circuits and conclude that, even in a timing-driven environment, critical paths are typically severely non-monotone; further we see that it is often the case that a relatively small fraction of the cells contribute to critical or near-critical path delays (thus indicating that a small amount of replication may have significant impact). Given these promising analyses, we propose an iterative algorithm that selectively replicates cells and perturbs the placement. The key components of this algorithm are as follows: (1) its incremental coupling with Static Timing Analysis to find a critical path (2) its notion of local monotonicity as a guide for replication decisions, the resulting notion of a feasibility zone and the careful placement of the cell within that zone (3) its careful attention to fanout partitioning to avoid degradation of other paths and (4) its legalization procedure to incrementally remove any cell overlaps induced by cell replication. As implied, we do not require that a replicated cell be placed in a currently vacant slot in the FPGA for this would be excessively limiting. Rather we let such cells be placed in near ideal slots regardless of the occupancy of such slots and use an incremental legalizer to resolve the overlaps with minimal change to the placement. Experimental results achieved from this approach are very promising with average improvements in critical path delay of 13% and over 25% on some large circuits.

2. MOTIVATION AND DEFINITIONS

2.1 An Example

A simple example illustrating the potential of replication with relocation is given in Figures 2 and 3. Suppose that cells A, B, D and E cannot be moved (maybe because they are pads or they are strongly connected to other blocks in their neighborhood). Cells B and D are inputs to this sub-network, cells A and E are output cells and cell C is intermediate. Thus, there are 4 distinct paths flowing through cell C. In this simple example assuming a linear delay model, minimization of the maximum path delay occurs with cell C placed in the center. Deviation from this point will degrade at least one of the paths. Thus, in this situation, this is the best we can hope for the placer to do. Also note that if the IO cells are in the corners of the unit square, the path lengths are 2 units for all 4 paths. On the other hand, Figure 3 shows what can happen if we duplicate cell C using one copy to drive cell A and the other to drive cell E. Once cell C has been replicated, we have more freedom in the placement of it and its copy. The result is that the placement can more easily be tuned to the needs of the various subpaths. In this case, we can obtain a dramatic improvement; in fact all 4 paths are roughly half the length as before and approach the lower-bound induced by the rectilinear distances between all input/output pairs – all of which are 1 unit. Also note that the total wire-length does not increase in the placement with replication.

Essentially, what has happened in this toy example is that replication allowed us to "straighten" non-monotone paths. From our initial experimental studies, we have seen that such non-monotonicity is frequently the limiting factor in clock period even for highly optimized designs (see Figure 1 for an example).

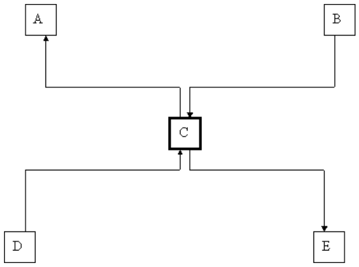

Figure 2: Example sub-circuit with timing-optimal placement of cell C. Assuming the other cells are fixed, any deviation from C's location will degrade at least one of the four paths.

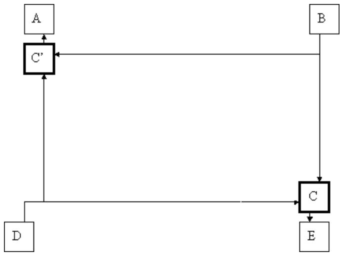

Figure 3: Optimized equivalent sub-circuit with replication of cell C and relocation.

2.2 Definitions

Throughout the paper we use the following terminology.

Path: sequence of interconnected modules with a starting point (input pad or memory element) and an ending point (output pad or memory element).

Critical Path: the path of the circuit with the largest total delay.

Criticality of a Module (or of an interconnection): ratio between the delay of the slowest path flowing through the module (or through the interconnection) and the current clock period.

Local Critical Sub Path: given a module, it's the sequence defined by the most critical preceding module the module itself and the most critical following module.

Monotone Local Sub Path: a sub path in which the distance between the first and the last module is equal to the distance between the first and the central module plus the distance between the central and the last module (monotone because the sequences of x and y coordinates are monotone sequences).

Monotone Region: given a sub path, it's the set of slots in which the central module of the sub path can be placed so that the sub path is monotone.

Arrival Time: given a pin, it's the time in which all the input signals are stable at the input pins of the module.

Downstream Time: given a pin, it's the maximum delay among all paths from that module to memory elements or output pads.

3. OBSERVATIONS

In this section we give background information on static timing analysis of placed circuits, reasonable delay models for FPGAs and some preliminary experiments which indicate high potential for logic replication.

3.1 Our Timing Analyzer and Delay Modeling

We have implemented a timing analyzer similar to that of VPR. The inputs of the timing analyzer are: connectivity of the modules of the circuit; placement of the modules; estimated pin-to-pin delay of each interconnection. The outputs of the timing analyzer are: arrival times of each module; downstream time of each module; critical path of the circuit, delays of the sinks on the circuits (output pads and memory elements). The complexity of the timing analyzer is linear. We implemented our own analyzer because it was easy to integrate with the replication algorithm.

In this study we consider a target architecture in which all the switches of the FPGA are buffered and interconnect resources are uniform (i.e., it is not a "segmented" architecture). With buffered switches, RC effects are localized to switch-to-switch connections; this has the effect of making segment delays independent of the rest of the net. Moreover, if the fanout of a net is low and the FPGA is not very congested, usually the router is able to route each interconnection on a monotone path. Thus the delay of an interconnection can be approximated by a linear function of the Manhattan length of the interconnection. Note, however that this linearity is not a property of the timing analyzer, but rather a property of the delay calculator that could be modified for other architectures.

3.2 Preliminary Analyses

Performing timing analysis on circuits placed by timing driven VPR has yielded three interesting observations. First, we studied the distribution of arrival times at sinks of the timing graph. What we see is that a relatively small number of sinks in the timing graph are near the critical path delay. This can be seen in Figure 4. This would seem to imply that optimization of a few critical sinks can have significant impact (though the number of paths to those sinks may still be large).

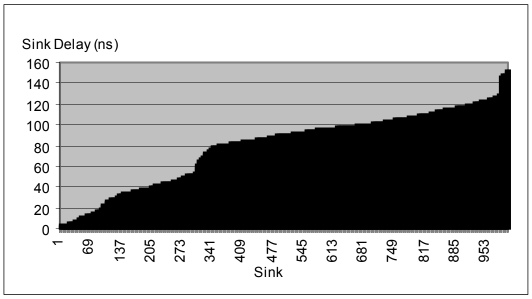

Figure 4: Sink delays of Frisc (sinks sorted by delay)

We draw a similar conclusion from a second set of experiments. For each individual cell in the netlist we can compute the maximum delay path through the cell. We are interested in how many such cells are near-critical. Suppose we say a cell is near-critical if the maximum delay through it is at least 90% of the critical path delay. It is interesting to know how many cells are near critical. Table 1 gives this data for several benchmarks. Only some 5% of the cells are near-critical and thus this would also seem to be a promising sign for a technique such as replication (e.g., the overhead may be manageable).

Table 1: Number of modules with criticality greater than 0.9

Circuit	*Number Cells*	*Number Critical*	*Percentage*
Frisc	3692	342	9.263272
Spla	3752	68	1.812367
s298	1941	186	9.582689
Elliptic	3849	189	4.910366
s38417	6541	150	2.293227
Des	2092	176	8.413002
bigkey	2133	13	0.60947
Average			**5.269199**

A third interesting piece of data is the physical locality of the near-critical cells. As shown in Figure 5, it is often the case that high criticality cells tend to be clustered. This suggests that our optimization procedure needs to be careful in such regions to avoid actually degrading performance. Our legalization procedure presented later in the paper is designed in part to help minimize such detrimental effects by being very incremental in nature.

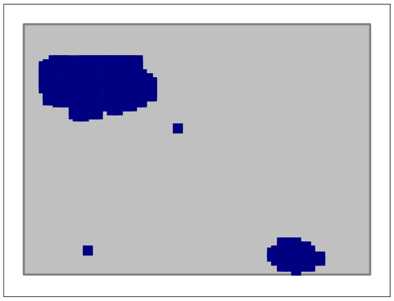

Figure 5: Positions of the nodes with criticality greater than 0.9 in Frisc

4. ALGORITHM DESCRIPTION

4.1 Overview

The algorithm, starting from an initial placement, iteratively generates a perturbation to the placement trying to relocate or duplicate a cell on the critical path in order to enforce monotonicity on the sub paths flowing through it. The algorithm chooses the new position for the cell taking into consideration only the timing constraints (a target clock period that must be reached) and not the physical constraints of the placement (cell overlapping). Possible overlaps are eliminated in a second legalization operation. The algorithm terminates when it fails getting an improvement for a certain number of iterations in row; the only accepted modifications are those that lead to a timing improvement of the circuit and the modifications are evaluated step by step. In Figure 6 there is the global view of the algorithm.

```
Replication () {
   Static_Timing_Analysis ()
       while (!Exit_Criterion()) {
           Old_Clock = Current_Clock
           while(Cells_Available&& !T_Constraints_Satisfied)
{
               Cell = Find_Cell ()
               Slot = Find_Slot (Cell)
               if (Target_Clock < Current_Clock) {
                   Timing_Constraints_Satisfied = TRUE
                   New_Cell = Generate_New_Cell (Slot)
                   Fanout_Partitioning (Cell, New_Cell)
                   if (Is_Overlap) {
                       Empty_Slot =Find_Empty_Slot()
                       Legalize (New_Cell, Empty_Slot) }
                   Static_Timing_Analysis ()   }            } }
           if (Current_Clock > Old_Clock)
               Restore_Old_Situation ()                     } }
```

Figure 6: Global View of the Algorithm

4.2 Cell Selection

The algorithm tries to improve the local monotonicity of the critical paths. Thus the cells that must be considered for optimization are those more greatly outside the monotone regions of the sub paths. The first intuitive idea is to take the cell that induces the greatest non-monotonicity on the sub paths that constitute the critical path. A measure of how much a cell is distant from its monotone region is its deviation. Deviation of a node i can be defined as:

$$deviation(i) = dist(prev(i), i) + dist(i, next(i)) - dist(prev(i), next(i))$$

The distance is intended as Manhattan distance and next(i) and prev(i) are the next and the preceding nodes on the critical path. This is the measure of how much a node is outside of its monotone region.

In order to choose the node, the algorithm evaluates for each node of the critical path its deviation from the monotone region. The vector is sorted by deviation from the biggest one to the smallest one. The selection of the cells will follow the order induced by the sorted vector. In order to make the algorithm non-deterministic the vector of the deviations is weighted with a random vector of values between 0 and 1. Then the resulting vector is sorted and gives the order for the cell selection. The algorithm can be made more or less deterministic manipulating the randomization routine. The reason why we make the algorithm non-deterministic is that with a randomized exploration of the solution space is possible a random restart scenario. We can run the algorithm more than once on the same circuit and then we can choose the best solutions. Experimental results confirm that the random restart framework is more effective than the simple greedy strategy.

4.3 Slot Selection

After cell selection the algorithm must find which is the best new placement for the cell.

The objectives that must be considered in this phase are to obtain the maximum performance gain on the critical path and to avoid that other paths flowing through the cell become more critical than the current critical path. The adopted routine is: generate a target clock; if, by relocation or replication of the selected cell, you can generate a new placement able to respect that maximum delay on all the paths flowing through the selected, accept that configuration; otherwise increase the target clock. The generated placements can violate the physical constraints because the legalizer will enforce them. If the procedure can't find a slot with a target clock better than the current clock, another cell is selected and the routine starts again. In Figure 7 there is the sub routine.

The Generate_New_Slot() routine generates candidate slots in breadth first order starting from the center of the monotone region defined by the sub path that has as its central cell the cell that is under consideration for replication.

```
Find_Slot (Cell) {
    Target_Clock=Generate_ Initial_Clock(Cell)
    while(Target_Clock<Current_Clock) {
        while (!All_Slots_Inspected && !Done) {
          Slot = Generate_New_Slot (Cell)
          Done=check_t_constraints(Cell,Slot, Target_Clock)  }
        if (Done)
          Return (Slot)
        else
          Target_Clock+=epsilon                              }
    Return (Failure)
```

Figure 7: Find Slot Routine

The Check_T_Constraints() routine checks if the selected slot satisfied the requirements of the target clock on all the sub paths that flow through the cell toward the critical output. It needn't to check also for the other outputs because they are automatically satisfied by the original slot and can only be improved by the new slot. The Check_T_Constraints() needn't to check the timing constraints for the non-critical successor because if they are not met for the replicated cell, they will be satisfied by the original one. It's here that the additional freedom degree of logic replication is exploited.

4.4 Fanout Partitioning

When we replicate a cell, we must decide from which copy each sink should receive the signal. This process is called fanout partitioning. There is a degenerate case in which all of the fanouts are assigned to only one of the copies; in this case, the other copy is redundant and reduces to simply relocating the original cell.

A naïve partitioning approach might simply connect each fanout to the nearest copy (note that the copy has been placed at this point). This approach is the wire length driven approach. However while this approach leads to a local wire length improvement, it can lead to timing performance degradation because it is oblivious to the requirements of paths flowing through the cells. An example appears in Figure 8. The dotted edges represent the two possible pin-to-pin connections. If you connect the cell 3 to the cell 2 you are reducing the wire length on that pin-to-pin interconnection; however, the path 1-2-3-4 is longer than the path 1-2R-3-4. Thus you are inducing degradation in timing performance.

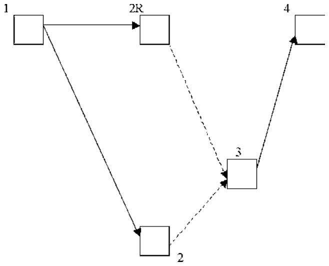

Figure 8: Failure scenario for wire length driver fanout partitioning

Our algorithm takes into account such a situation. It checks which is the best timing configuration for each pin-to-pin interconnection and select the best one. If at the end the old cell has 0 fanout, it is deleted.

4.5 Legalization

The purpose of the legalizer is, starting from an illegal placement (with overlap of modules), to obtain a placement without any overlap. There is a danger that perturbations done by a legalizer may degrade the performance of other paths. If you consider that the critical nodes are usually very close to each other, the degradation of near critical paths seems to be a common situation during legalization. In our algorithm legalization occur after each placement perturbation and so we must only solve one violation cell at a time.

Our approach tries to limit degradation by inducing only small modifications to the current placement. Toward this end, we adopt a "ripple-move" approach similar to that in [8]. In such an approach we identify the overlap slot and a vacant slot somewhere in the region. By a sequence of cell movements from the overlap slot toward the vacant slot, we can resolve the violation. Further, note that no cell moves more than one slot in such an approach; the incremental nature would seem to limit the amount of degradation on other paths. Notice also that there may be many possible ripple sequences. We choose the sequence which gives the maximum gain in wire-length (which may be negative). This is achieved by finding a longest path in a gain-graph as in [8].

Figure 9 illustrates such a gain graph. For the computation of the wire length gain the model that is used is the source-to-sink length model because is better for timing purposes. Edge labels are wire-length gains for the associated move. Though we currently use wire-length as the guide, we note that other edge labels may be useful (e.g., those relating to timing).

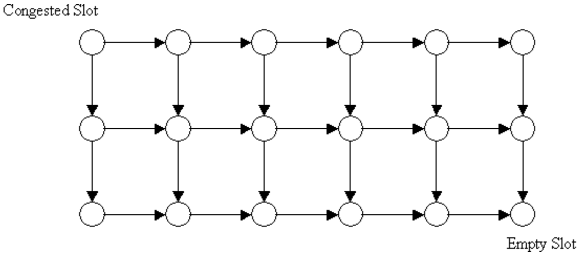

Figure 9: Cost Graph for Legalization

5. EXPERIMENTAL RESULTS

Table 2 shows the experimental results for 20 MCNC benchmark circuits. The circuits are placed using VPR in timing driven mode. Then they are optimized with our algorithm and routed using VPR in timing driven mode. The data that we compare are the maximum delay through the circuit (as reported by VPR) after and before optimization by replication, the total routed wire length and the number of logic blocks. The circuits were placed on the minimum square FPGA able to contain the circuit and routed with a number of tracks per channel that is 20% more than the minimum required in order to route the circuit. The experimental framework is the same described in [9] in order to test the timing driven placer of VPR. The circuits are divided in two sets: the smallest ones (less than 3000 blocks) and the largest ones (more than 3000 blocks). Our algorithm works better with the largest ones. In fact as you can see the average timing improvement is 15.29% on the largest circuits and 11.69% on the smallest circuits. At the same time the wire length degradation and the amount of replication is smaller for the biggest circuit. We believe this to be a promising trend as typical circuits tend toward the larger side. These results are the best of three runs of our algorithm. The run time overhead introduced by our algorithm to the timing driven placer of VPR is less than 0.5%.

The reader may notice that on two smaller circuits there is actually a small degradation in delay. This can be understood by the fact that our timing optimizer is pre-routing and thus there is bound to be some deviation from this estimate post-routing. This is also confirmed by the fact that pre-routing timing analysis reports a timing improvement that is on average 1% better than the post-routing timing analysis. Fortunately, post-routing degradations seem to be small and the placement-level predictors behave quite well in general.

6. CONCLUSIONS

We have presented a preliminary study of placement-level logic replication with relocation applied to FPGA timing optimization. We developed a path-based algorithm that exploits replication trying to optimize the local monotonicity of the most critical paths. Experimental results are encouraging showing consistent improvement in critical path delay, particularly as circuit size increases; reductions in critical path delay of up to 29% were observed.

Table 2

	Circuit	Before Timing Optimization			After Timing Optimization			Percentage Modification		
		Worst Delay	WL	Blocks	Worst Delay	WL	Blocks	Worst Delay	WL	Blocks
SMALLEST	ex5p	111.51	18656	1135	85.17	20197	1146	23.63%	7.63%	0.96%
	tseng	76.34	9143	1221	69.77	9201	1228	8.61%	0.63%	0.57%
	apex4	113.62	22143	1290	99.29	22155	1312	12.61%	0.05%	1.68%
	misex3	105.15	21989	1425	85.11	22700	1439	19.06%	3.13%	0.97%
	alu4	120.35	20785	1544	97.82	22106	1553	18.72%	5.98%	0.58%
	diffeq	80.86	14963	1600	74.80	15164	1605	7.49%	1.33%	0.31%
	dsip	74.93	15311	1796	74.97	16454	1797	-0.05%	6.95%	0.06%
	seq	117.96	28148	1826	95.73	29281	1840	18.84%	3.87%	0.76%
	apex2	132.42	31080	1919	99.58	33369	1936	24.80%	6.86%	0.88%
	s298	130.58	21910	1941	130.93	22855	1946	-0.27%	4.13%	0.26%
	des	82.93	27244	2092	82.32	28291	2092	0.73%	3.70%	0.00%
	bigkey	69.01	20667	2133	64.83	21432	2134	6.06%	3.57%	0.05%
	AVERAGE							**11.69%**	**3.99%**	**0.59%**
LARGEST	frisc	154.64	60119	3692	124.86	60617	3700	19.26%	0.82%	0.22%
	spla	129.69	67278	3752	126.78	69356	3778	2.24%	3.00%	0.69%
	elliptic	112.08	49123	3849	107.90	49738	3857	3.73%	1.24%	0.21%
	ex1010	212.68	71165	4618	173.84	73177	4644	18.26%	2.75%	0.56%
	pdc	255.05	102557	4631	179.45	107980	4680	29.64%	5.02%	1.05%
	s38417	123.01	65621	6541	123.01	65621	6541	0.00%	0.00%	0.00%
	s38584_1	122.92	54952	6789	95.62	55207	6791	22.21%	0.46%	0.03%
	clma	239.85	138244	8527	175.20	145167	8583	26.95%	4.77%	0.65%
	AVERAGE							**15.29%**	**2.26%**	**0.43%**

7. REFERENCES

[1] L.T. Liu, M.T. Kuo, C.K. Cheng, and T.C. Hu. A Replication Cut for Two-Way Partitioning, IEEE Trans. on CAD, 1995

[2] J. Hwang and A. El Gamal. Optimal replication for min-cut partitioning. ICCAD 1992

[3] W. K. Mak and D. F. Wong. Minimum replication min-cut partitioning. IEEE Transactions on CAD,. October 1997

[4] C. Kring and A. Newton. A cell-Replicating Approach to Mincut-Based Circuit Partitioning. ICCAD 1991

[5] V. Betz and J. Rose.VPR: A New Packing, Placement and Routing Tool for FPGA Research. 7th International Workshop on Field-Programmable Logic and Applications, 1997

[6] J. Lillis, C.-K. Cheng, T.-T. Y. Lin. Algorithms for Optimal Introduction of Redundant Logic for Timing and Area Optimization. Proc. IEEE International Symposium on Circuits and Systems, 1996

[7] A. Srivastava, R. Kastner, M. Sarrafzadeh.Timing Driven Gate Duplication: Complexity Issues and Algorithms. ICCAD 2000

[8] S.-W. Hur and J. Lillis.Mongrel: Hybrid techniques for standard cell placement. ICCAD 2000.

[9] A. Marquardt, V. Betz and J. Rose. Timing-Driven Placement for FPGAs. International Symposium on FPGAs, 2000

[10] W. Gosti, A. Narayan, R. K. Brayton, A. L. Sangiovanni-Vincentelli. Wireplanning in logic synthesis. ICCAD 1998

[11] W. Gosti, S. P. Khatri, A. L. Sangiovanni-Vincentelli. Addressing The Timing Closure Problem By Integrating Logic Optimization And Placement. ICCAD-2001

[12] Neumann, D. Stoffel, H. Hartje, W. Kunz. Cell Replication and Redundancy Elimination During Placement for Cycle Time Optimization. ICCAD 1999