Re-engineering of Timing Constrained Placements for Regular Architectures *

Anmol Mathur
Dept. of Computer Science
U. of Illinois, Urbana-Champaign
Urbana, IL 61801

K. C. Chen
Fujitsu Labs of America
77 Rio Robles
San Jose, CA 95134

C. L. Liu
Dept. of Computer Science
U. of Illinois, Urbana-Champaign
Urbana, IL 61801

Abstract

In a typical design flow, the design may be altered slightly several times after the initial design cycle according to minor changes in the design specification either as a result of design debugging or as a result of changes in engineering requirements. These modifications are usually local and are referred to as engineering changes. *In this paper we study the problem of* timing driven placement re-engineering *: the problem of altering the placement of a circuit to incorporate engineering changes without degrading the timing performance of the circuit. We focus on the re-engineering problem for regular architectures such as FPGAs and gate arrays. Our algorithms exploit the locality of the re-engineering design changes and use the current placement to generate the new placement for the altered circuit. Our experiments on the Xilinx 3000 FPGA architecture demonstrate the effectiveness of our algorithm in handling engineering changes efficiently.*

1 Introduction

Minor changes in the design specification after the initial design flow are referred to as *engineering changes*. The process of generating a new design that incorporates the modifications to the circuit is referred to as *design re-engineering*. Specifically, we are interested in re-engineering a placement under timing constraints : a problem we call *timing driven placement re-engineering*. After the initial design cycle, a completely placed and routed design that satisfies all the timing constraints is available. The input to the problem of timing driven placement re-engineering is a placed and routed design and the engineering changes made to the circuit. The output is a new layout that incorporates the design changes and satisfies all the (possibly new) timing constraints. Since the engineering changes are typically local and relatively minor, it would be wasteful to re-do the entire layout of the modified circuit from scratch. Further, the localized nature of engineering changes means that the layout of the original circuit is a good starting point for finding a delay feasible layout of the modified circuit. An algorithm for minimizing the perturbation in the placement while handling engineering changes was presented in [2]. Techniques have also been proposed for exploiting the locality of engineering changes in circuit simulation [4]. More recently there has been an effort to integrate logic synthesis and layout through the use of local logic restructuring driven by routability considerations [1]. Such a technique for integration of logic synthesis and layout would benefit greatly from the use of an efficient algorithm for timing driven placement re-engineering.

In this paper, we study the problem of timing driven placement re-engineering for *regular architectures*. The two architectures of interest are Field Programmable Gate Arrays (FPGAs) and gate arrays. The concept of a slack neighborhood graph is used to design efficient an efficient algorithm for reconfiguring a physically infeasible placement (with some slots occupied by two modules) to a feasible one with provably low timing degradation. This serves as a crucial part of our algorithm for the timing driven placement re-engineering problem. Our algorithm exploits the regularity of the underlying architecture to ensure that the execution time is significantly less than that for re-doing the placement from scratch for the modified circuit. The regularity of these architectures allows us to use powerful graph theoretic techniques that would not be possible otherwise.

The organization of the remainder of the paper is as follows: Section 2 is devoted to a more precise formulation of the placement re-engineering problem; Section 3 describes our algorithm for placement re-engineering; and Section 4 presents the results of our experiments with the Xilinx 3000 series FPGA architecture.

2 Placement Re-engineering

After an iteration of the design flow, we have a placed and routed design I that will be referred to as the *current implementation*. The *current circuit specification* is $P = \langle C, T \rangle$, where C represents the current circuit topology and T is the set of *timing constraints* that must be satisfied by a *delay feasible* implementation.

- C is represented by a directed graph $G_C(V_C, E_C)$ where each module of the circuit is represented by a vertex in V_C and an interconnection between two modules is represented by an edge in E_C between the corresponding vertices.

*Work partially supported by NSF under grant MIP 92-22408 and by Fujitsu Laboratories of America, Inc.

- The timing constraints consist of arrival times $A(x_i)$ at each primary input of the circuit, the module delay, $d(M_i)$, for each circuit module and the required time, $R(f_i)$, by which the signal should arrive at primary output f_i.

The circuits considered in this paper could be either combinational or sequential, however for the purpose of timing calculations we assume that the circuit has been cut at all the latches (thus creating pseudo-primary inputs and outputs). We assume that T contains information about the arrival times and required times at these pseudo primary inputs and outputs. Thus, the timing calculation algorithms will assume that the underlying graph is a directed *acyclic* graph.

In *placement re-engineering* we are given the current circuit specification, P, the current implementation, I, and some engineering changes. The output of a placement re-engineering algorithm is a delay feasible placement for the new specification, P'. Further, it is desirable that the running time of the algorithm be proportional to the number of engineering changes.This makes it necessary to make use of I in generating the new delay feasible placement for P'.

2.1 Engineering Changes

Typically, engineering changes have the following characteristics:

1. **Locality:** Engineering changes result from local restructuring of the circuit. So, they do not completely change the structure or function of the circuit. Changes that have global effect on the timing behavior or functionality of the circuit will not be classified as engineering changes.
2. **Small size:** The *zone of influence* of a change can be defined as the portions of the circuit that are affected by the change. Re-engineering changes should have a zone of influence that is a small fraction of the entire circuit. Since it is hard to estimate the zone of influence of a change, we will assume that the number of gates and nets that get changed by a set of engineering changes is a small fraction of the size of the circuit.

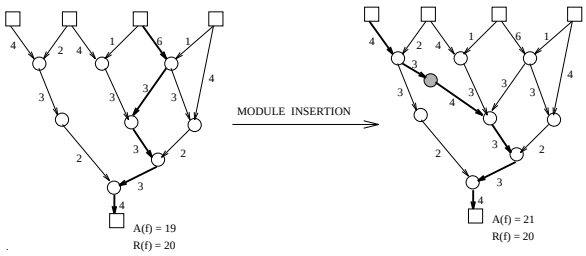

Fig. 1: Structural changes can result in violation of timing constraints. Addition of the shaded module results in a new critical path (dark path) that increases the arrival time at the output from 19 to 21 causing violation of the required time constraint.

Engineering changes can be classified into two broad categories depending on which component of the specification tuple $P\langle C, T\rangle$ is changed.

- **Structural changes:** These changes alter the structure of the circuit and result in changes to the circuit topology. The structural changes that we consider are:
  1. *Module Addition:* Addition of new circuit modules along with the associated interconnections.
  2. *Module Deletion:* Deletion of some of the existing modules and their connections to other modules.
  3. *Interconnect Addition:* Addition of new interconnections among existing modules.
  4. *Interconnect Deletion:* Deletion of interconnections between circuit modules.
- **Timing changes:** These changes alter the timing constraints imposed on any delay feasible implementation of the circuit. We consider the following two kinds of timing changes:
  1. *Change in arrival time at a primary input or required time at a primary output.*
  2. *Change in delay of a circuit module.*

Notice that the addition of new modules and interconnections can cause violation of timing constraints, even if there is no change in the timing constraints, due to the creation of new paths in the circuit (see Fig. 1). When a module or interconnection is deleted from the circuit, the implementation of the circuit can be easily updated to reflect the change by simply removing some geometries from the layout. Since this will not increase the arrival time of any signal, the resulting layout is delay feasible. So, from the point of view of timing driven placement re-engineering, deletions are trivial to handle. In this paper, we will henceforth focus only on structural changes involving the addition of modules and/or interconnections and timing changes.

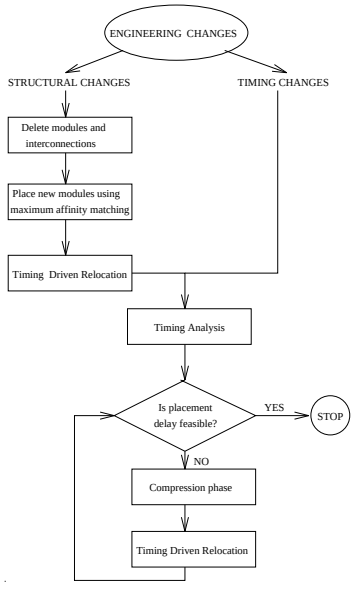

Fig. 2: An overview of our approach to timing driven placement re-engineering.

3 Our Algorithm

Figure 2 gives an overview of our approach to timing driven placement re-engineering. For structural changes, first we remove the modules and interconnections to be deleted from the placement. For the new modules being added to the circuit, we have an initial placement phase followed by an iterative compression-relaxation phase if the initial placement is not delay feasible. For timing changes, we perform timing analysis with the modified constraints and if the current placement is not delay feasible the iterative compression-relaxation scheme is used.

Our algorithm for generating the initial placement for new modules added as part of structural changes to the circuit is characterized by the use of a physically infeasible intermediate placement in the process of generating a new placement. We make use of an efficient algorithm for timing driven relocation in transforming a physically infeasible placement that has some slots occupied by two circuit modules to a physically feasible one with low timing degradation. The concept of a slack neighborhood graph is introduced and demonstrated to be useful in guaranteeing low timing degradation during the movement of modules in timing driven relocation. Timing driven relocation is also used in the relaxation phase of the compression-relaxation iterations.

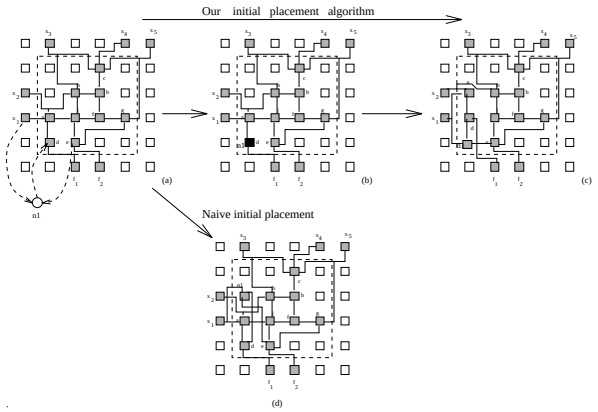

Fig. 3: Comparison of our initial placement algorithm with a naïve one that places the new module in an empty slot close to the neighbors of the new module. (a) A placed and routed design to which a new module, $n1$, is being added. (b) A placement is determined for the new module using maximum affinity matching creating an overlap (shown as the dark slot). (c) New placement after timing driven relocation to remove the overlap. (d) The new placement generated by an algorithm that tries to find an empty slot near the modules to which the new module has high connectivity.

In order to clarify the problem of timing driven placement re-engineering and motivate our approach, let us consider an example of an engineering change where a new module is being added to a circuit that has already been placed. Figure 3 shows an example of a new module being added to a circuit that is already placed and routed. Figures 3 (a) – (c) show the steps involved in our algorithm for handling module insertion. Figure 3(d) shows the new placement produced by a naïve algorithm that simply searches for a vacant slot in the vicinity of the slots to which the new module has high connectivity. Notice that if a naïve algorithm is used for the placement of the new module, the new module is placed in a slot that is quite far from the modules to which it has high connectivity. This is because the slots in the vicinity of the new module's neighbors are occupied. On the other hand, our algorithm allows the new module to be placed on an occupied slot in the initial placement phase and then performs a timing driven relocation of the modules to remove overlaps.

Our algorithm for the initial placement of new modules involves an **affinity based initial placement** of the new modules. Affinity based placement attempts to minimize the timing degradation during the placement of the new modules, but it may place the new modules in slots that are already occupied. Consequently, the placement may be physically infeasible after affinity based placement. **Timing driven relocation** is used to reconfigure such a physically infeasible intermediate placement to a physically feasible one, without significant timing degradation.

The above approach tries to minimize the timing degradation in the process of addition of new modules, but does not guarantee that the new placement will meet all the timing constraints. If the designer is willing to tolerate some timing degradation in the process of re-engineering then this approach is fine. However, if the new placement must be delay feasible then we need to use the **compression-relaxation approach** [5].

3.1 Affinity Based Placement

The first phase of our algorithm for handling the addition of new modules along with their interconnections involves selecting a good initial placement for the new modules. At this stage we allow the new modules to be placed in slots that are already occupied, thus resulting in a physically infeasible placement. Allowing a physically infeasible intermediate placement results in a significant increase in the quality of the final placement (see Fig. 3).

We denote the *affinity* of a module M_i to slot S_j by $A(M_i, S_j)$. Intuitively, the affinity of a new module to a slot is a measure of the quality of the new placement that results when the module is placed in that slot. Both the increase in the total wire length and the total increase in the arrival times of signals at the primary outputs are taken into consideration in the computation of the affinities. If there is a large increase in the total wire length or the total of the arrival times of the signals at the primary outputs by placing the new module M_i at slot S_j, then $A(M_i, S_j)$ is low. If a new module is placed in a slot that is not occupied, then it will not force any relocation of modules in the second phase of our algorithm. The affinity function takes this into consideration by having a term that depends on the status (vacant or occupied) of the slot. Thus, the affinity is defined by the following weighted sum

$$A(M_i, S_j) = \alpha \cdot (\Delta L(M_i, S_j))^{-1} + \beta \cdot (\Delta T(M_i, S_j))^{-1} + \gamma \cdot \mu(S_j),$$

where

$\Delta L(M_i, S_j)$ = total increase in the lengths of all the nets when module M_i is placed on slot S_j,

$\Delta T(M_i, S_j)$ = total increase in the arrival times of signals at all the primary outputs of the circuit when module M_i is placed on slot S_j,

$$\mu(S_j) = \begin{cases} 1 & \text{if } S_j \text{ is vacant} \\ 0 & \text{if } S_j \text{ is occupied} \end{cases}$$

Notice that the definitions of $\Delta L(M_i, S_j)$ and $\Delta T(M_i, S_j)$ assume that M_i is the only new module being added to the circuit and compute the effect of placing it on slot S_j on the total wire length and total arrival time respectively. The effect of the other new modules being added to the circuit are ignored in the calculation of $A(M_i, S_j)$. Any attempt to simultaneously capture the effect of adding several modules to the circuit results in a drastic increase in the time required for computing the affinities. The computation of ΔL takes time proportional to the number of nets incident to the module for while the affinity is being computed. The computation of ΔT requires time proportional to the size of the fanout cone of the new module. Thus, the total time consumed in the computation of the affinities is $O(k \cdot (|E_C| + |V_C|) \cdot n)$ where n is the number of slots in the architecture and $|E_C| + |V_C|$ is the size of the circuit graph.

We use the affinities computed above to place the new modules in slots in the underlying architecture. This is done by finding a *maximum matching* in a weighted bipartite graph, $G(V_M \cup V_S, E)$ where V_M is the set of vertices corresponding to the new modules being added to the circuit, V_S is the set of vertices corresponding to the slots in the architecture. The edges in E are of the form (u, v), $u \in V_M$, $v \in V_S$. Further, each edge $(u, v) \in E$ has a weight equal to the affinity of the module represented by vertex u to the slot represented by vertex v. Since we allow for the possibility of a module being placed in any of the slots, the graph G is a *complete* bipartite graph. So a maximum matching of size $|V_M|$ always exists (assuming $|V_M| < |V_S|$). Our algorithm for generating the initial placement of the new modules finds a maximum weight matching in G using mincost flow in a modified graph having unit node capacities. Such a matching yields a placement of the new modules, with the module corresponding to vertex $u \in V_M$ being placed on the slot corresponding to vertex $v \in V_S$ if edge (u, v) is in the maximum weight matching. Since we have computed a maximum weight matching the resulting initial placement maximizes the sum of the affinities of the new modules to the slots on which they get placed. At the same time, no two new modules are placed on the same slot. This ensures that in the placement that results, a slot is occupied by at most two modules.

Thus, after initial placement of the new modules, we end up with a placement that may be physically infeasible. If the placement is physically infeasible we need to relocate the old modules occupying the "overcrowded" slots to vacant slots. Further, in the process of relocation it is essential to ensure that timing degradation is minimized. This is accomplished by using *timing driven relocation.*

3.2 Timing Driven Relocation

The input to the timing driven relocation problem is the modified circuit specification, along with a physically infeasible placement generated by the initial placement step. The set of slots occupied by two modules constitutes the set of *infeasible* slots. Timing driven relocation moves the circuit modules occupying infeasible slots to feasible slots through series of moves ending at vacant slots. Further, the relocation is done such that the critical path delay of the circuit does not increase significantly. An obvious necessary condition for relocation to be possible is that there should be at least as many vacant slots as there are infeasible ones.

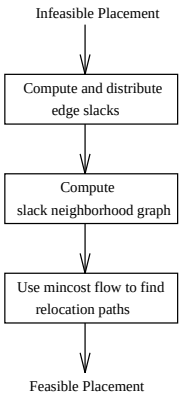

Fig. 4: The main steps in our algorithm for timing driven relocation.

Figure 4 shows the main steps in our algorithm for timing driven relocation using the concept of a **slack neighborhood graph**. Using the placed and routed design, we compute the slack of each edge in the circuit graph which is a measure of the amount by which the delay on the edge can be increased without violating any of the timing constraints. Since a delay increase can be translated into an increase in the length of the interconnection corresponding to the edge, the slack can be interpreted as an upper bound on the amount by which the length of the interconnection can be increased, without violating the timing constraints.

Informally, the set of neighboring slots to which a module can be moved without violating any timing constraints is said to be its *slack neighborhood.* The graph in which the adjacency relation reflects these slack neighborhoods is referred to as the *slack neighborhood graph.* Details regarding the construction of the slack neighborhood graph and its use in developing a provably good algorithm for timing driven relocation (refer to [5]) are omitted due to lack of space.

3.3 Handling Timing Violations

Once we have a physically feasible placement for the modified circuit we perform timing analysis using the new timing specification. If the current placement violates some timing constraints, we use an iterative algorithm based on the compression-relaxation approach of [5] to produce a delay feasible placement. This section provides a brief overview of the compression-relaxation approach. Details can be found in [5].

The compression phase attempts to make the placement delay feasible by compressing the long paths that cause some of the primary output signals to arrive too late. However, the compression phase may produce an infeasible placement with some of the slots occupied by two (but no more than two) modules. This allows the compression

phase more flexibility and often allows it to achieve the required decrease in delay. If an infeasible placement is produced in the compression phase, the relaxation phase, which carries out a *timing driven relocation* to produce a physically feasible placement. The slack neighborhood graph based algorithm for timing driven relocation described in Section 5 is used in the relaxation phase.

A delay analysis is carried out on the current placement to compute the actual arrival times of the signals at the various outputs and the outputs where the timing requirement is violated are identified. Associated with each such output is a set of long paths ending at the output, that are responsible for the violation of the timing requirement. We define the *longest path tree* (LPT) of a primary output, f_i, to be the subgraph of the cone of f_i consisting of the edges in the longest path from each vertex in the cone of f_i to the primary output f_i. The LPT captures several longest paths in the cone of f_i simultaneously. The compression algorithm attempts to satisfy the timing requirement at f_i by moving modules in the cone to new slots so as to decrease edge delays, a process referred to as *compressing the cone at* f_i. It should be noted that in the process of compressing $cone(f_i)$, up to two modules can be placed in the same slot. This ensures that compressing of the cone has enough flexibility to guarantee a substantial decrease in the critical path delay of the cone. If there are several outputs at which the arrival time exceeds the required time, the cone of the output with the maximum violation of the required time is chosen for compression. Details of the construction of the LPT and the compression algorithm can be found in [5].

The compression algorithm attempts to make the task of the relaxation phase easier by relocating modules to empty slots, if they are available at the right distance to achieve the required compression. Further, an attempt is made to spread out the infeasible slots to make the relocation in the relaxation phase easier.

4 Experimental Results

We have implemented our algorithms as a package called *reeng*. For our experiments we used the Xilinx 3000 series FPGA architecture as the underlying architecture on which circuits are placed. Experiments were conducted using technology mapped MCNC combinational benchmark circuits.

The initial delay feasible placement for the circuits was generated using the *sysdias* package [5]. Structural and timing changes were generated randomly. Both the number of structural and timing changes and the circuit modules affected by the changes were controlled probabilistically. Roughly speaking, the expected number of engineering changes of a particular type introduced by our algorithm for generating random engineering changes is equal to the product of the probability of that change and the number of nodes in the circuit. We studied the behavior of our algorithm with the number of changes made to the circuit specification.

One of the crucial requirements for any placement re-engineering algorithm is that it should be significantly faster than re-doing the placement for the modified circuit from scratch. In order to study this aspect of our algorithm, we compare the number of compression-relaxation iterations required to generate a delay feasible placement and the run time for *reeng* with those required by *sysdias* for placing the modified circuit from scratch. Since *sysdias* starts from a randomly generated initial placement and then performs compression-relaxation iterations until a delay feasible placement is obtained, the number of compression-relaxation iterations used by *reeng* and *sysdias* is a good measure of the complexity of placement re-engineering using these two methods. These results are shown in Table 1. The probabilities for structural changes and timing changes were both set to 0.1 for the generation of engineering changes. Thus, roughly 10% of the nodes and interconnections were affected by the structural changes and around 10% of the timing constraints were altered. The amount by which a timing constraint was tightened (that is the reduction in required time, $R(x)$) in a timing change was a uniformly distributed random variable in the range $[0, 0.1R(x)]$. The columns SC and TC refer to the number of structural and timing changes respectively. In SC only module additions/deletions are counted. Table 1 shows that the run time for *reeng* is an order of magnitude smaller than the run time for *sysdias* and the number of compression-relaxation iterations is also proportionally smaller. This confirms our hypothesis that the delay feasible placement for the modified circuit is "close" (in the configuration space of all placements) to the delay feasible placement for the original circuit and hence can be found more efficiently by using the delay feasible placement for the original circuit as a starting point. Table 1 also shows the actual delay in the critical path for the placed circuits after routing using the Xilinx *apr* tool. The critical path delays using *reeng* and using *sysdias* are approximately the same. This is to be expected since both the packages terminate once all the timing constraints are satisfied.

We also studied the variation in run time of *reeng* with respect to the probability of structural and timing changes. The graph in Figure 5 shows the variation in the run time for *reeng* on circuit *alu2*. It is interesting to observe that the increase in running time with probability of structural change is almost monotonic. The non-monotonic sections of the graph may be due to the fact that it is possible that a smaller number of changes may sometimes have a larger affect on the critical paths in the circuit. Also a higher probability of change does not necessarily imply that the actual number of changes introduced by our algorithm for generating random engineering changes is larger. The increase in run rime is slightly faster when the probability of timing change is increased. This could be due to the fact that timing changes tend to have a more global effect on the placement since they might force the relocation of several modules. Structural changes on the other hand are more local, and only if there is a violation of some timing constraint due to a structural change are its effects propagated to other parts of the circuit.

Another parameter of interest in evaluating a placement re-engineering algorithm is the difference between the initial placement and the new placement generated after placement re-engineering. We measured the variation in the number of module movements during re-engineering with the probability of change. We observed that there is a strong correlation between the running time of *reeng* and the number of module movements. Figure 6 shows the variation in number of module movements with structural change probability for *alu2*.

Circuit	# mod	EC		*sysdias*			*reeng*		
		SC	TC	# iter	Time	Delay	#iter	Time	Delay
cordic	48	6	4	25	48	50.5	4	3	51.0
count	85	7	10	32	85	60.0	3	3	58.5
bw	61	9	5	20	80	32.0	4	4	30.0
f51m	58	7	5	21	93	83.4	3	3	85.0
frg1	91	12	8	32	150	75.5	5	6	75.0
comp	103	10	12	40	260	80.4	4	4	82.0
term1	161	18	16	45	303	70.6	5	6	75.0
C499	144	12	10	33	260	73.0	4	4	72.6
C880	210	18	22	42	335	120.5	5	6	125.0
alu2	232	21	22	48	410	183.5	7	7	185.4

Table 1: Comparison of running time (in seconds), delay (after routing) of the critical path (in nanoseconds) and number of compression-relaxation iterations for *reeng* and *sysdias*. In *sysdias* the entire placement is re-done from scratch.

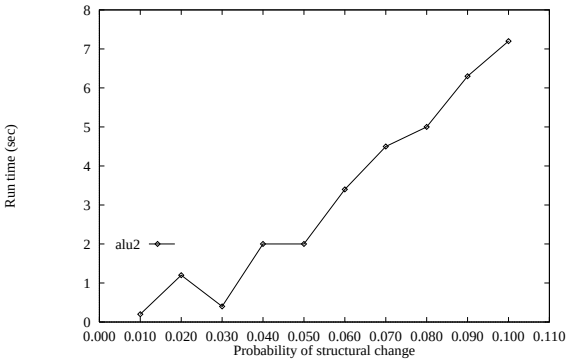

Fig. 5: Variation in run time of *reeng* with increase in structural changes.

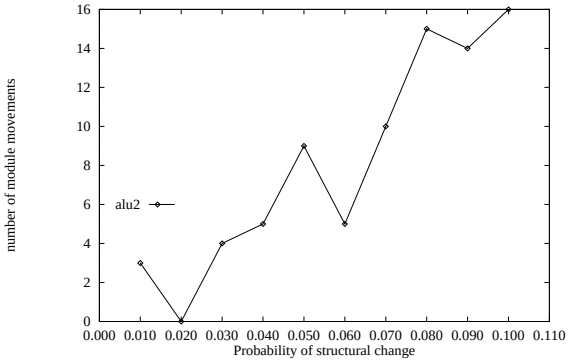

Fig. 6: Variation in the number of module movements with structural change probability.

5 Conclusions

We have presented a new approach to the timing driven placement re-engineering problem for regular architectures. Our algorithm handles both structural changes in the circuit topology as well as changes in the timing constraints imposed on the placement. A novel algorithm for generating the initial placement for new modules using maximum affinity matching is proposed. Timing driven relocation using the concept of a slack neighborhood graph is used in transforming physically infeasible intermediate placements to physically feasible ones with low timing degradation. Our experimental results on the Xilinx 3000 series FPGA architecture demonstrate the viability of our approach.

References

[1] S. C. Chang, K. T. Cheng, N. S. Woo, M. Marek-Sadowska, Layout Driven Logic Synthesis, *Proc. DAC, 1994, pp. 308–313.*

[2] C. Choy, T. Cheung, An Algorithm to Deal with Incremental Layout Alteration, *Proc. 34th Midwest Symposium on Circuits and Systems, Vol. 2, 1991, pp. 850–853.*

[3] B. Codenotti, R. Tamassia, A Network Flow Approach to the Reconfiguration of VLSI Arrays, *IEEE Transactions on Computers, 40 (1991), pp. 118–121.*

[4] Y. Ju, Incremental Circuit Simulation and Timing Analysis Techniques, *PhD. Thesis, Univ. of Illinois at Urbana-Champaign, 1993.*

[5] A. Mathur, C. L. Liu, Compression-Relaxation: A New Approach to Performance Driven Placement for Regular Architectures, *Proc. ICCAD, 1994, pp. 130–136.*

[6] A. Mathur, K. C. Chen, C. L. Liu, Applications of Slack Neighborhood Graphs to Timing Driven Optimization Problems in FPGAs, *Proc. 3rd ACM/SIGDA Symposium on FPGAs, 1995, pp. 118–124.*