OPTIMALITY AND STABILITY STUDY OF TIMING-DRIVEN PLACEMENT ALGORITHMS

Jason Cong, Michail Romesis, Min Xie

Computer Science Department
University of California, Los Angeles
{cong,michail,xie}@cs.ucla.edu

ABSTRACT

This work studies the optimality and stability of timing-driven placement algorithms. The contributions of this work include two parts: 1) We develop an algorithm for generating synthetic examples with known optimal delay for timing driven placement (T-PEKO). The examples generated by our algorithm can closely match the characteristics of real circuits. 2) Using these synthetic examples with known optimal solutions, we studied the optimality of several timing-driven placement algorithms for FPGAs by comparing their solutions with the optimal solutions, and their stability by varying the number of longest paths in the examples. Our study shows that with a single longest path, the delay produced by these algorithms is from 10% to 18% longer than the optima on the average, and from 34% to 53% longer in the worst case. Furthermore, their solution quality deteriorates as the number of longest paths increases. For examples with more than 5 longest paths, their delay is from 23% to 35% longer than the optima on the average, and is from 41% to 48% longer in the worst case.

1. INTRODUCTION

Placement is one of the most important steps in the post-RTL synthesis process as it directly defines the interconnects, which have now become the bottleneck in circuit and system performance in DSM technologies. The placement problem has been studied extensively in the past 30 years. However, a recent study shows that existing placement solutions are surprisingly far from optimal. Using a set of constructed placement examples that match many industrial circuit characteristics with known optimal wirelength (PEKO), the study shows that the results of leading placement tools from both industry and academia are 70% to 150% away from the optimal solutions on those examples [1]. An extension of PEKO was presented in [2], where new examples called PEKU (Placement Examples with Known Upper bounds) were created by inserting a certain percentage of non-local nets into a PEKO circuit. By relaxing the optimality constraint on a subset of connections, PEKU more accurately emulates real circuits in terms of wirelength distribution. Experiments showed that for PEKU benchmarks, state-of-the-art placers can be far away from the upper bound. In the extreme case, where each circuit consists of global connections only (G-PEKU benchmarks), existing tools can be 41% to 102% away in the worst case. These studies generated great interest in both academia and industry.

However, wirelength is not the sole objective in circuit placement. In the era of DSM technology, an important goal of placement is performance (delay) optimization. There is a strong need to extend the optimality study to timing-driven placement algorithms.

Existing timing-driven placement algorithms can be divided into two categories, net-based and path-based. Path-based algorithms [3, 4, 5] try to directly minimize the longest path delay. Since they maintain an accurate timing view during optimization, their complexity is usually high. Net-based algorithms [6, 7, 8, 9] first transform timing constraints into either length constraints or weights on individual nets. The information is fed to a weighted wirelength minimization based placement engine to obtain a new placement with better timing. This process usually goes through multiple iterations until no improvement can be made, or a certain iteration limit has been reached. Compared with path-based algorthms, net-based algorithms usually have lower complexity.

There are several works on generating timing-driven placement examples [10, 11]. However, none of them satisfy our need, since their optimal solutions are unknown. In this paper, we present an algorithm for generating timing-driven placement examples with known optimal delay under a simplified delay model (T-PEKO). These examples can closely match the characteristics of real circuits. Using these examples with known optimal delays, we studied the optimality of several timing-driven placement algorithms for FPGAs from commercial and academic tools by comparing their solutions to the optimal solutions, and their stability by varying the number of longest paths in the examples. We chose FPGA placement since it gives the flexibility to specify our delay model and cell library.

Experimental results for the academic tools show that for examples with a single longest path, the delay produced by the algorithms is from 10% to 18% longer than the optima on average, and from 34% to 53% longer in the worst case. Furthermore, their solution quality deteriorates as the number of longest paths increases. For examples with more than 5 longest paths, their delay is from 23% to 35% longer than the optima on average, and is from 41% to 48% longer in the worst case. The performance of the commercial tool that targets the Xilinx Virtex architecture is much smaller. The difference in delay from our constructed solution, on average, is 8% without routing and 4% after routing. To our knowledge, this is the first study on the optimality of timing-driven placement algorithms.

The rest of this paper is organized as follows: Section 2 presents the T-PEKO algorithm for the construction of timing-driven placement examples with a known optimal solution. Section 3 presents the comparison of the placement results for the T-PEKO suite produced by state-of-the-art, timing-driven placement algorithms with the optimal solutions. Section 4 presents conclusions and future

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

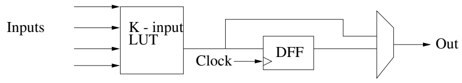


Figure 1: Graph of a basic logic element. It consists of a lookup table (LUT), a flip-flop and a multiplexer

work.

2. CONSTRUCTION OF PLACEMENT EXAMPLES WITH KNOWN TIMING OPTIMAL SOLUTION

2.1. Discussion of the FPGA Architecture and the Delay Model

We first present the architecture of the FPGA device that we assume during the construction of the examples. Each logic block (CLB) consists of two basic logic elements (BLE). A BLE is shown in Figure 1, and it contains a K-input LUT, a flip-flop and a multiplexer. The flip-flop's input is connected to the output of the LUT. The multiplexer selects the output of the LUT or the flip-flop.

Several delay models have been proposed to calculate the performance of a circuit. The most popular is the Elmore delay model [12]. Recent studies, e.g., [13], have shown that under optimal buffer insertion, sizing and wire sizing, the delay of a wire is approximately linear to its length. For this reason, in this paper we use a linear delay model which can be summarized as follows:

(i) The delay inside any LUT is a constant d_g, while any other delay inside a BLE is assumed to be zero.

(ii) The delay of any interconnect $d_i(A, B)$ between two BLEs A and B is given by the following formula: $d_i(A, B) = dist(A, B) * d_u$, where $dist(A, B)$ is the Manhattan distance of BLE A from BLE B, while d_u is the constant delay between two adjacent BLEs (Manhattan distance equal to 1).

In reality, the BLEs of FPGA devices are more complex and include more connections, as will be shown in the Xilinx experiment section. The delay model also can be more complicated. However, our methodology is generic in that it is applicable as long as the interconnect delay between two adjacent nodes is always smaller than any delays between non-adjacent nodes and d_g, d_u are constants. Furthermore, it can be applied to ASICs as well, especially to standard-cell row-based architectures.

2.2. The T-PEKO Algorithm

Our methodology for the construction of the timing-optimal benchmarks works as follows: The first step is to obtain a placement solution of an existing combinational or sequential circuit. The second step is to perform timing analysis to find the longest path in the circuit using our delay model. Let d_k be the delay of the longest path, and w, h be the number of rows and columns of the device, respectively. The algorithm perturbs the netlist by inserting a path p_{opt} that connects $r+1$ adjacent nodes, where r is computed by $r = max(\lceil d_k/(d_g + d_u)\rceil, \lceil ((w+h) * d_u + d_g)/(d_g + d_u)\rceil)$ (see Figure 2). Since the new netlist is the result of a perturbation of the original netlist, the smaller the perturbation, the stronger the similarities it has with the original circuit.

Before we present in detail the construction of the path p_{opt}, we denote some terms here:

- We call a netlist *valid* if: 1) It has no combinational loops, 2) It has no dangling BLEs, i.e., BLEs with at least one input (output) and no outputs (inputs), and 3) Each BLE has at most K inputs and 1 output. If a netlist is not valid, it is called *invalid*.
- *Static timing analysis* [14] constructs a *timing graph* whose vertices correspond to the pins of the circuit. The *timing edges* that connect the vertices of this graph are constructed in two ways: 1) Each net is converted into a set of directed edges that connect each source of the net to all sinks of the net. 2) Each LUT is represented by a set of *intracellular* edges that connect all the inputs of the LUT to its output.
- Each LUT is assigned a number called the *Level* of the LUT, such that the following property is satisfied:

  Property (1): For every timing edge of the timing graph of the circuit originating from the output pin of an LUT a to an input pin of another LUT b, we have $Level(a) < Level(b)$.

  It is easy to see that if this property is satisfied, there are no combinational loops in the circuit. If some timing edges violate the above property, we can guarantee that by removing them the circuit is free of combinational loops. Note that this property does not have to hold for timing edges between pins of the same LUT, or between pins of a LUT and a flip-flop.
- A flip-flop f is *unused* if the multiplexer of the BLE selects the output of the LUT l, otherwise it is *used*. In the remainder of this paper, when we say that the status of a flip-flop is changed from used to unused, we perform the following changes to the netlist: The flip-flop is removed from the netlist and all the fanouts of the flip-flop become fanouts of the LUT except for the ones that cause a violation of Property (1). Similarly, when we mention that the status of a flip-flop is changed from unused to used the following changes are performed: A new net is added to the netlist from the output of the LUT to the input of the flip-flop, and all the previous fanouts of the LUT become fanouts of the flip-flop.

Before the construction of the path p_{opt} these initial steps take place:

(i) The original mapped netlist is placed on the FPGA device.

(ii) Static timing analysis is performed on the placed circuit. The longest path delay d_k is computed as well as the integer r according to the formula :

  $r = max(\lceil d_k/(d_g + d_u)\rceil, \lceil ((w + h) * d_u + d_g)/(d_g + d_u)\rceil)$. The first expression calculates the number of BLEs required by d_k. The second expression guarantees the delay of the longest path is no less than the delay between any BLE and IO pad. This is because our algorithm may connect a dangling BLE with an arbitrary IO pad, as will be explained later in this section.

  Every LUT is assigned to a level equal to the highest arrival time among its pins.

The construction of the path p_{opt} is as follows:

(i) A BLE is selected at a corner of the device as the first node of the path. If the flip-flop of the BLE is unused its status is changed to used. A new timing edge (if it does not exist already) is added [1] from the output pin of that flip-flop to an input pin of an adjacent LUT, which we call the *current* LUT of the path. Then, we check if the input constraint of the current LUT of the path is violated. If the current LUT had all its K input pins used before the addition of the new edge, the algorithm randomly selects one of them and removes the timing edge [2] that corresponds to this pin. After removing a timing edge, it is possible that another BLE becomes dangling. This is fixed by the following process: We find the closest BLE to the dangling one that has its flip-flop used and at least one unused input (if the dangling node does not have outputs) and connect the dangling node to it. If a feasible BLE cannot be found, a PO at the boundary of the circuit is randomly selected and connected to the dangling node. Although this process can increase the number of IOs, it does so only slightly as the experimental results will show.

Note that the BLEs corresponding to the two LUTs may be initially unused in the placement solution. If this is the case, the LUTs and the used flip-flop will be added to the netlist.

(ii) The following procedure is repeated $r - 1$ times:

A new timing edge is added (if it does not exist already) connecting the output pin of the current LUT a of the path to an input pin of an adjacent LUT b, as in the previous case. The selection of the adjacent LUT is such that the path has a snake shape (see Figure 2), in order to guarantee that all the LUTs can be visited exactly once. LUT b becomes the current LUT of the path, and the flip-flop in the same BLE is changed to unused. If this change causes a BLE to become dangling (because some connections are removed if Property (1) is violated), the algorithm performs the same steps as in (i). The algorithm will check if the input constraint of the current LUT is violated and, if so, will fix it in the same way as in (i). Furthermore, it will check if Property (1) is violated. If it is violated, we will have $Level(a) \geq Level(b)$. The violation is fixed by reassigning the level of LUT b and by removing some timing edges if necessary. We divide the fanouts of b into two groups: those that belong to LUTs with level higher than $Level(a)$, and those that belong to LUTs with level equal to or lower than $Level(a)$. These two sets are denoted as S_1 and S_2 respectively. All the timing edges from b to S_2 will be removed. Let the LUT with the lowest level in S_1 be c. Level(b) will be assigned as $(Level(a) + Level(c))/2$. If the set S_1 is empty, $Level(b)$ can be assigned to any value higher than Level(a). It is obvious that after these changes, Property (1) is satisfied. Some nodes may become dangling because their inputs are removed. The algorithm will either connect them with the output of BLEs whose flip-flops are used or PIs, similar to that in step (i).

(iii) At the end of step (ii), the *current* LUT is the last node of the path p_{opt}. If its corresponding flip-flop is used, it is changed to used.

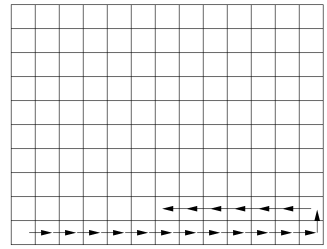


Figure 2: Example of an artificial longest path. It starts from a corner of the device and has a snake shape in order to guarantee that all the nodes can be visited exactly once

Path p_{opt} connects $r + 1$ adjacent BLEs. It goes through r LUTs and r connections between adjacent BLEs, so the total delay of p_{opt} is $TD = r * (d_g + d_u) \geq d_k$. If this path is the longest of the circuit for this placement, it is obvious that TD is the optimal delay of the circuit for the given LUT mapping and delay model.

However, the addition of new timing edges and the changes on the flip-flops during the construction of p_{opt} may create some new paths that have longer delays than TD. In order to shorten these paths, we iteratively perform timing analysis on the perturbed circuit until the delay of the circuit becomes equal to TD. If the timing analysis shows that the longest path's delay TD' is longer than TD, we identify the critical path, and remove the interconnect timing edges along that path that are not in p_{opt}. During this process, we will also fix dangling BLEs, as in the previous step. Eventually our artificial path will become the longest of the circuit, as we will prove later.

The following theorem states the validity of the algorithm:

Theorem 1 : The T-PEKO algorithm guarantees that the perturbed netlist is valid, and that p_{opt} is the longest path of the placed netlist.

2.3. Increasing the Difficulty of the T-PEKO examples

In this work we study not only the optimality of the timing-driven placement algorithms, but also their stability for circuits with different characteristics. Ideally a stable algorithm is expected to perform well on various kinds of circuits. For this stability study we introduce two parameters for the construction of the circuits that control their difficulty for a placer, including:

(i) *The number of longest paths* : The algorithm can create a user-specified number of disjoint longest paths. Assume that this number is M, that the delay of the critical path of the original circuit is d_t, and that integer r is computed as before. We create a path p_{opt} according to the same methodology as described earlier with the only difference that it connnects $M * r + 1$ adjacent BLEs. The total delay of that path will be $TD = M * r * (d_g + d_u) \geq M * d_t$. Along this path at equal distances, we insert $M - 1$ flip-flops. As a result, the initial longest path is replaced by M paths, each one with a delay greater than or equal to d_t. Note that after this change, some other paths in the circuit might become longer, and they will be removed according to the same procedure as described in the previous subsection. In the end, these M paths will become the longest of the circuit.

[1] The insertion of a timing edge on the timing graph from the output pin s of LUT a to an input pin t of LUT b corresponds to the following changes on the netlist: If pin s is used and n is the corresponding net, add pin t to the sinks of n. If s is not used, create a new 2-pin net with s as its source and t as its sink.

[2] The removal of a timing edge from the output pin s of LUT a to an input pin t of LUT b corresponds to the following changes on the netlist:If the corresponding net n has more than 2 pins, remove pin t from its sinks. If n has only 2 pins, remove the net from the netlist.

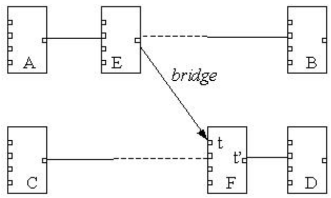


Figure 3: Bridge construction. A bridge will be inserted between s and t.

(ii) *The number of edges that connect longest paths* : To increase the degree of path sharing, T-PEKO will create some nets to connect BLEs located on different longest paths. Figure 3 provides an example. AB and CD are two longest paths constructed as described in the previous paragraph. E is a BLE along the path AB, F is a BLE along the path CD. T-PEKO will connect E's output s with one of F's unused inputs t. This corresponds to inserting a timing edge between s and t in the timing graph. We call the newly added timing edge a *bridge*, denoted as $b(s,t)$. The following theorem guarantees that the netlist remains valid after this operation.

Theorem 2: The netlist after inserting $b(s,t)$ is valid if $d(s) + dist(E,F) * d_u + d_g \leq d(t')$. Here, $d(s)$ is the arrival time of s, t' is the output pin of F, $d(t')$ is the arrival time of t', and dist(E,F) is the Manhattan distance of E from F. The longest paths in the original netlist remain the longest after $b(s,t)$ is inserted.

2.4. Extension to the Xilinx Architecture

The previously described algorithm targets our simplified model and FPGA architecture. With some modifications, T-PEKO is extended to create placement examples constructed for commercial tools. More specifically, in this subsection we describe how we created examples for the Xilinx Virtex architecture. In this architecture a CLB (configurable logic block) contains two slices, and each slice contains 2 LUTs (see Figure 4). Due to the interconnect architecture of Virtex [15], it is not guaranteed that the interconnect delay between adjacent nodes is shorter than the delay between two non-adjacent nodes.

The artificial path we constructed for this architecture was slightly different from the general case of the previous section. The path will first visit all four nodes of a CLB before moving to

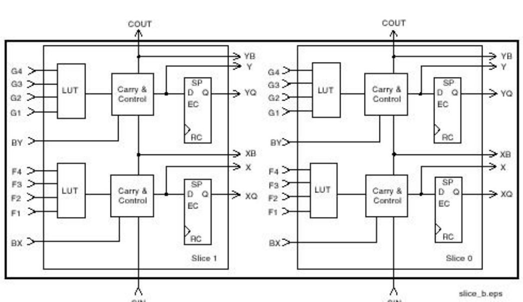


Figure 4: A Virtex CLB contains 4 LUTs in 2 slices. Picture taken from the web site of Xilinx

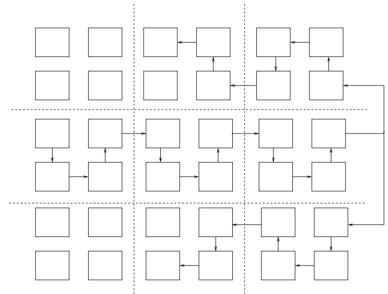


Figure 5: An example of the artificial path on a Xilinx Virtex device. A box represents an LUT, while the dashed lines show the borders between different CLBs. The path traverses all the LUTs of a CLB, before moving to an adjacent CLB.

an adjacent CLB. Figure 5 shows an example of two artificial paths that we created for a Xilinx Virtex device. These paths share the same CLBs in the middle row, but the first path moves to the CLBs of the upper row, while the other path moves to the CLBs of the bottom row. For our Xilinx experiments, we used this technique to create multiple paths. Similar changes must be performed when working on other FPGA architectures.

One additional problem is that the delay model is no longer known. It is true that delay tables can be extracted [3], but they may not be 100% accurate. Therefore, the timing analysis we perform is an approximation. It is not guaranteed that the artificial path is the longest path in the circuit. Still, we can consider the delay of that path as an upper bound of the optimal delay of the circuit.

In the experimental results section we shall investigate the performance of the Xilinx place and route tool PAR on the T-PEKO examples tailored for the Virtex architecture.

3. EXPERIMENTAL RESULTS

We implemented T-PEKO on a Sun Blade 1000 using C++. To generate the initial placement configurations needed by T-PEKO, we ran VPR [16] on 20 MCNC benchmarks using its timing driven mode. The placement results were then fed into T-PEKO and perturbed. We varied M from 1 to 5 and generated 100 circuits. The maximum number of inputs and outputs on each BLE is 6 and 1 respectively. As for the delay parameters, $d_g = 1$ and $d_u = 1$. When possible, a maximum of 50 bridges were inserted between the M longest paths. We call these circuits the T-PEKO suite and make them available at [17].

Table 1 gives the characteristics of T-PEKO in terms of the number of CLBs, PIs, POs, flip-flops and nets. The column "Orig" shows the name of the original MCNC circuit from which the initial placement configuration is derived. The columns for $M = 0$ show the characteristics of the original MCNC circuits. Column "Opt" gives the optimal delay under our simplified delay model. For the same initial placement configuration, the optimal delay does not change for any value of $M > 0$ (of course, we do not know the optimal delay for $M = 0$). The perturbed circuits are very close to the original ones in these aspects for most cases. The

[3]We built the delay tables in the Virtex architecture as follows: A net connecting 2 LUTs is constructed. One LUT was fixed at a corner of the chip. The other was moved to every location on the chip. We filled the delay table with the delays reported by the Xilinx timing analysis tool in every case.

circuits that were initially combinational were transformed into sequential after the insertion of flip-flops (40 in the worst case on these circuits). The circuits are given in the format specified in [18]. Each circuit has a .net file describing the netlist of each circuit. It also has a .arch file specifying the combinational delay of each LUT, and the number of IOs for each CLB. To guarantee a fair comparison, we generated a .pad file for each circuit, which gives the pad locations extracted from the optimal solutions by our construction. The functions and formats of .arch .net and .pad files are specified in [19].

For our optimality and stability study, we experimented with two state-of-the-art FPGA placement algorithms, including:

- VPR [16], a well-known FPGA placement and routing package widely used for FPGA architecture evaluation [19]. Its optimization engine is based on simulated annealing. It combines connection-based and path-based timing-analysis. The cost function it uses trades off between wirelength and critical path delay. We used VPR v.4.3 downloaded from [20] in our experiment.
- PATH [21], the latest FPGA placement algorithm which presents a significant enhancement to VPR in timing optimization. It takes into consideration the path sharing effect. PATH introduces a new net weighting algorithm based on the concept of path-counting. We used PATH v.1.0 in our experiment.

One complication of FPGA architecture is that the delay between two BLEs depends not only on their Manhattan distance, but also the routing segments that connect the BLEs. Therefore, both algorithms use a preliminary routing procedure before placement to determine the delay between BLEs. To accommodate our simplified delay model, we modified the delay computation in each algorithm, so that the delay between BLEs is always the Manhattan distance between them multiplied by d_u. This change, in effect, makes our study of these algorithms independent of the FPGA architecture and their routing procedure. In our experiment, we set the tradeoff parameter of wirelength vs. delay to be 0.5, as suggested by [16]. Changing the value of this parameter to favor the critical path delay minimization did not seem to improve the final results.

For each circuit of T-PEKO, we run each algorithm 5 times. The results are summarized in Table 2 and Figure 6. The average difference between each algorithm's result and the optimal solution is listed. For completeness, the best results for every circuit are reported. From the results, we make the following observations:

- For $M = 1$, the delay produced by the algorithms is from 10% to 18% longer than the optima of T-PEKO on average, and from 34% to 53% longer in the worst case.
- The solution quality of both algorithms deteriorates as M increases. For $M = 5$, the gap between their solutions and the optima is from 23% to 35% on average, and from 41% to 48% in the worst case.
- PATH outperforms VPR in all cases. The best results from PATH are on average 4% worse than the optima when $M = 1$, and 18% worse when $M = 5$.

Figure 7 shows the optimal configuration of TPeko20 with $M = 5$ and the results generated by both VPR and PATH. The nodes

Table 2: Experimental results by VPR and PATH on the TPEKO suite. M correponds to the number of initial longest paths. Average and minimum divergence from the optima by VPR and PATH is listed.

Circuit	Opt	M = 1				M = 3				M = 5			
		VPR		PATH		VPR		PATH		VPR		PATH	
		Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best
TPeko01	88	4%	0%	7%	0%	21%	17%	17%	15%	35%	27%	25%	20%
TPeko02	108	6%	1%	4%	1%	23%	15%	17%	13%	38%	31%	16%	14%
TPeko03	106	12%	4%	6%	0%	30%	23%	12%	8%	27%	24%	15%	10%
TPeko04	116	7%	0%	0%	0%	10%	7%	5%	3%	18%	14%	15%	9%
TPeko05	92	22%	7%	10%	0%	34%	25%	15%	5%	34%	24%	16%	11%
TPeko06	84	10%	5%	5%	1%	26%	20%	19%	8%	34%	20%	17%	14%
TPeko07	100	22%	10%	4%	0%	25%	20%	11%	7%	33%	27%	17%	13%
TPeko08	132	53%	44%	18%	7%	52%	47%	30%	24%	40%	20%	33%	27%
TPeko09	124	9%	2%	0%	0%	19%	14%	24%	15%	29%	28%	21%	18%
TPeko10	122	11%	6%	7%	1%	27%	25%	18%	12%	33%	29%	15%	13%
TPeko11	128	15%	10%	8%	0%	26%	19%	10%	6%	40%	33%	15%	12%
TPeko12	166	11%	4%	4%	1%	40%	16%	13%	11%	39%	15%	16%	14%
TPeko13	170	31%	25%	15%	8%	39%	22%	35%	19%	35%	29%	32%	22%
TPeko14	146	12%	8%	2%	1%	26%	21%	21%	12%	32%	26%	31%	25%
TPeko15	184	15%	11%	7%	1%	25%	22%	17%	10%	36%	33%	23%	18%
TPeko16	240	17%	7%	13%	3%	61%	55%	14%	12%	36%	25%	19%	13%
TPeko17	290	24%	17%	10%	2%	24%	15%	17%	12%	30%	21%	26%	21%
TPeko18	164	32%	15%	19%	5%	33%	15%	30%	20%	29%	22%	30%	16%
TPeko19	164	17%	13%	34%	26%	46%	25%	45%	38%	48%	41%	41%	37%
TPeko20	328	32%	25%	24%	14%	49%	37%	25%	19%	47%	30%	38%	31%
Avg.		18%	11%	10%	4%	32%	23%	20%	13%	35%	26%	23%	18%

on the longest paths by our construction are colored in black in each solution. Furthermore, the critical timing edges in each solution are also colored in black. It can be seen that these nodes are indeed on the longest paths of both VPR and PATH's results. However, the delay produced by both algorithms is far away from the optimal. Note that besides the longest path created by T-PEKO, there exist some other paths with the same delay, that include nets from the original circuit. Figure 7 shows several such paths in the optimal solution.

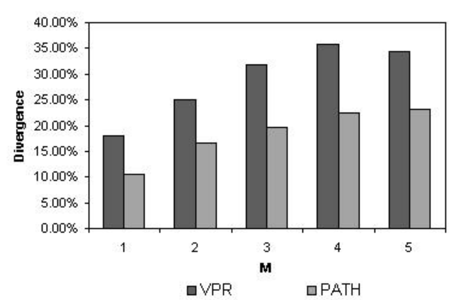


Figure 6: Divergence vs M.

Using the method described in the previous section, we extended our study to the Xilinx placement engine, PAR, and constructed 17 synthetic circuits from MCNC benchmarks [4]. The version we experimented is Release 5.1.03i - PAR F.26. First, we let PAR do placement without routing and compared the delay with that of the constructed solutions. Then we let PAR do placement

[4]The TPEKO-generated circuit for bigkey runs out of pads, since the initial placement of bigkey has a high number of IO pads very close to available pads on the chip. The initial solution of s38417 and s38584.1 include some active flip-flops that are not connected to the LUT in the same BLE, which is not compatible TPeko's assumption.

Table 1: Characteristics of the TPeko suite. Column “Orig” gives the initial circuit from which the perturbed circuits are derived. $M = 0$ corresponds to the characteristics of the original circuit. The perturbed circuits are very close to the original circuits in the number of CLBs, PIs, POs, flip-flops and nets. Column “Opt” gives the optimal delay for each circuit. It is the same for circuits derived from the same initial placement for $M \geq 1$.

Ckt	Orig	Opt	M = 0					M = 1					M = 3					M = 5				
			CLB	PI	PO	FF	NET	CLB	PI	PO	FF	NET	CLB	PI	PO	FF	NET	CLB	PI	PO	FF	NET
TPeko01	tseng	88	1047	51	122	385	1098	1056	51	105	371	1107	1059	51	85	341	1128	1060	51	81	326	1162
TPeko02	ex5p	108	1064	8	63	0	1072	1067	8	57	5	1075	1068	8	56	6	1102	1068	8	50	32	1127
TPeko03	apex4	106	1261	9	19	0	1271	1275	9	22	25	1284	1287	9	17	24	1310	1287	9	18	38	1347
TPeko04	dsip	116	1370	228	197	224	1598	1424	228	192	226	1652	1537	228	189	228	1789	1653	228	189	230	1932
TPeko05	misex3	92	1397	14	14	0	1411	1415	14	34	40	1429	1427	14	15	19	1461	1431	14	11	23	1496
TPeko06	diffeq	84	1497	63	39	377	1560	1502	63	26	366	1565	1511	63	19	354	1596	1512	63	18	343	1626
TPeko07	alu4	100	1522	14	8	0	1536	1535	14	8	2	1549	1546	14	8	6	1579	1561	14	8	11	1626
TPeko08	des	132	1591	256	245	0	1847	1604	255	206	10	1859	1686	254	194	16	1971	1772	254	191	17	2077
TPeko09	bigkey	124	1707	228	197	224	1935	1766	228	197	226	1994	1793	224	188	239	2036	1801	223	166	227	2075
TPeko10	seq	122	1750	41	35	0	1791	1754	41	40	18	1795	1756	41	38	27	1814	1756	41	28	24	1848
TPeko11	apex2	128	1878	38	3	0	1916	1894	38	8	13	1932	1912	38	5	17	1977	1913	38	8	25	2002
TPeko12	s298	166	1931	3	6	8	1934	1934	3	6	11	1937	1934	3	6	15	1986	1934	3	6	42	1988
TPeko13	frisc	170	3556	19	116	886	3575	3568	19	113	896	3587	3576	19	109	893	3628	3578	19	105	876	3648
TPeko14	elliptic	146	3604	130	114	1122	3734	3616	130	81	1099	3746	3628	130	68	1054	3789	3638	130	63	1027	3819
TPeko15	spla	184	3690	16	46	0	3706	3698	16	50	14	3714	3706	16	47	25	3773	3706	16	53	35	3773
TPeko16	pdc	240	4575	16	40	0	4591	4595	16	55	30	4611	4607	16	48	35	4667	4607	16	39	27	4674
TPeko17	ex1010	290	4598	10	10	0	4608	4610	10	21	22	4620	4612	10	10	14	4673	4612	10	10	21	4673
TPeko18	s38417	164	6406	28	106	1463	6434	6417	28	96	1477	6445	6434	28	94	1485	6501	6441	28	88	1435	6520
TPeko19	s38584.1	164	6435	37	304	1260	6484	6457	37	262	1250	6494	6476	37	236	1236	6552	6487	37	233	1211	6575
TPeko20	clma	328	8382	61	82	33	8444	8393	60	60	128	8453	8402	60	57	117	8513	8408	60	53	120	8519

Table 3: Experimental results on Xilinx PAR.

Circuit	chip/package	IOB	Slice	Net	w/o routing			w routing		
					UB	PAR	diff	UB	PAR	diff
TPeko01-x	xcv50/bg256	73	573	1130	70.0	75.2	7.5%	76.8	80.4	4.7%
TPeko02-x	xcv200/fg456	176	582	1182	78.2	80.3	2.7%	90.6	90.5	-0.1%
TPeko03-x	xcv50/bg256	33	669	1302	62.7	65.8	4.9%	64.7	67.5	4.3%
TPeko04-x	xcv50/bg256	31	760	1415	55.9	62.5	11.9%	54.7	56.9	4.1%
TPeko05-x	xcv50/bg256	27	766	1537	52.2	57.4	10.0%	49.0	53.5	9.2%
Tpeko06-x	Xcv50/bg256	110	766	1563	59.2	61.2	3.5%	55.9	58.3	4.2%
TPeko07-x	xcv600/fg680	435	832	1690	78.2	83.5	6.7%	90.7	95.0	4.7%
TPeko08-x	xcv600/fg680	504	863	1932	78.2	83.0	6.1%	90.2	91.1	1.1%
TPeko09-x	xcv100/bg256	79	941	1832	64.6	68.1	5.5%	68.3	69.9	2.4%
TPeko10-x	xcv100/bg256	49	968	1940	60.7	66.0	8.6%	61.7	64.4	4.3%
TPeko11-x	xcv100/bg256	16	1199	1964	64.8	67.2	3.7%	65.1	67.9	4.3%
TPeko12-x	xcv200/fg456	152	1824	3562	57.9	63.8	10.1%	57.4	61.3	6.8%
TPeko13-x	xcv200/fg456	62	1961	3731	61.8	66.1	7.0%	62.8	65.6	4.5%
TPeko14-x	xcv200/fg456	248	1985	3766	62.7	71.3	13.7%	65.0	69.4	6.8%
TPeko15-x	xcv200/fg456	59	2350	4593	53.2	61.7	15.9%	50.5	53.5	6.0%
TPeko16-x	xcv200/fg456	29	2350	4614	55.2	63.6	15.1%	53.1	55.6	4.8%
TPeko17-x	xcv600/fg680	130	4704	8463	64.3	69.8	8.6%	68.5	67.3	-1.7%
Avg.							8.3%			4.1%

followed by routing. In the latter case, we used PAR to do routing on our constructed solutions and quoted the delay reported by its timing analysis tool. The delay on our constructed solutions served as upper bounds to the optimal delay for the synthetic circuits. To guarantee that PAR can find the minimum possible delay in this experiment, we set a loose delay constraint at the beginning and gradually tighten it until PAR can no longer find a solution satisfying this constraint.

Table 3 gives the experimental results on these circuits. The first few columns give the circuit characteristics. The upper bound of the optimal delay by our construction is given in the column “UB,” the result by PAR is given in the column “PAR.” The delays are given in nano-seconds. On average, the delay generated by PAR is 8.3% worse than our constructed solutions without routing, and 4.1% after routing. Compared with our experiment with VPR and PATH, the divergence here is much smaller, especially after routing. In fact, for some cases, the result by PAR is better than our constructed solution. One possible reason is that the delay between two elements on a Virtex chip is not monotone with regard to their Manhattan distance. It depends heavily on the routing path chosen for each net.

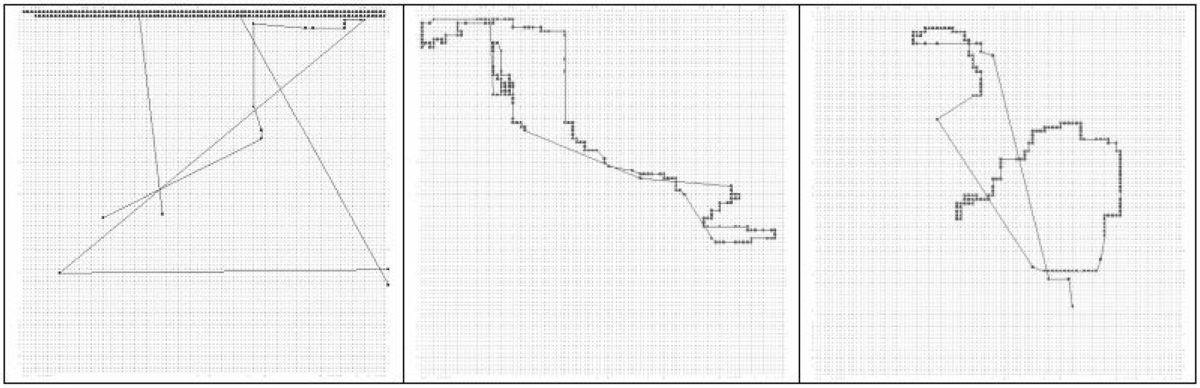
4. CONCLUSIONS AND FUTURE WORK

This work studied the optimality and stability of timing-driven placement algorithms. We developed an algorithm for generating synthetic examples with known optimal delay for timing driven placement (T-PEKO). The synthetic examples generated by our algorithm can closely match the characteristics of real circuits. Using these synthetic examples with known optimal solutions, we studied the optimality of several timing-driven placement algorithms by comparing their solutions to the optimal solutions, and their stability by varying the number of longest paths in the examples. The results produced by the algorithms could be as far as 54% away from the optimal for our most difficult examples. The results seem to suggest that timing-driven placement algorithms, both net-based and path-based, have room for improvement. The performance of the commercial tool that targets the Xilinx Virtex architecture is much smaller. The difference in delay from our constructed solution, on average, is 8% without routing and 4% after routing.

Future work includes the generation of similar placement examples that study the performance of placement algorithms for other objectives such as routability and power on both ASIC and FPGA designs.

5. ACKNOWLEDGEMENTS

This work is partially supported by the Semiconductor Research Corporation under Contract 98-TJ-686 and 2003-TJ-1019, and partially supported by the National Science Foundation under Grant CCR0096383. The authors would like to thank Dr. T. Kong from Magma for providing the latest version of PATH for our experiments. They would like to thank Dr. R. Jayaraman from Xilinx for his valuable suggestions regarding the Xilinx experiments. They would also like to thank A. Jagannathan and Y. Lin for their help



Optimal solution VPR's solution PATH's solution

Figure 7: Three solutions for TPeko20. The nodes on the longest paths by our construction are colored in black. The timing edges on critital paths in each solution are colored in black, too. It can be seen these nodes are indeed on the longest paths in both VPR and PATH's results. However, the delay produced by both algorithms are far away from the optima. Note that besides the longest paths created by T-PEKO, there exist other paths with the same delay, that include nets from the original circuit. Several of them are shown in the optimal solution.

on the experiment with Xilinx tools.

6. REFERENCES

[1] C. Chang, J. Cong, and M. Xie, "Optimality and scalability study of existing placement algorithms," in *Proc. Asia South Pacific Design Automation Conference*, pp. 621 – 627, 2003.

[2] J. Cong, M. Romesis, and M. Xie, "Optimality, scalability and stability study of partitioning and placement algorithms," in *Proc. International Symposium on Physical Design*, pp. 88 – 94, 2003.

[3] M. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. Design Automation Conf*, pp. 370–375, 1989.

[4] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement for small-cell ICs," in *Proc. Int. Conf. on Computer Aided Design*, pp. 48–51, 1991.

[5] T. Hamada, C. K. Cheng, and P. M. Chau, "Prime: a timing-driven placement tool using a piecewise linear resistive network approach," in *Proc. Design Automation Conf*, pp. 531–536, 1993.

[6] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proc. Design Automation Conf*, pp. 133–136, 1984.

[7] R. Nair, C. L. Berman, P. Hauge, and E. J. Yoffa, "Generation of performance constraints for layout," *IEEE Trans. on Computer-Aided Design*, vol. 8, no. 8, pp. 860–874, 1989.

[8] R. S. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," in *Proc. Design Automation Conf*, pp. 620–625, 1991.

[9] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. Design Automation Conf*, pp. 269–274, 1998.

[10] M. Hutton, J. P. Grossman, J. Rose, and D. Corneil, "Characterization and parameterized random generation of digital circuits," in *Proc. Design Automation Conf*, pp. 94–99, ACM Press, 1996.

[11] P. Verplaetse, D. Stroobandt, and J. Van Campenhout, "Synthetic benchmark circuits for timing-driven physical design applications," in *Proc. International Conference on VLSI*, pp. 31–37, CSREA Press, 2002.

[12] W. C. Elmore, "The Transient Response of Damped Linear Networks," *Journal of Applied Physics*, vol. 19, pp. 55 –63, 1948.

[13] J. Cong, and D. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Asia Pacific Design Automation Conference*, pp. 97–100, 1999.

[14] R. Hitchcock, G. Smith, and D. Cheng, "Timing Analysis of Computer Hardware," *IBM J. Res. Develop.*, vol. 26, pp. 100–108, 1982.

[15] Xilinx Inc., *Virtex 2.5V FPGA Complete Data Sheet (all four Modules)*.

[16] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. of the ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 203–213, ACM Press, 2000.

[17] http://cadlab.cs.ucla.edu/∼pubbench/tpeko.htm.

[18] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *Proc. of Seventh International Workshop on Field-Programmable Logic and Applications*, pp. 213–222, 1997.

[19] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[20] http://www.eecg.toronto.edu/∼vaghn/vpr/vpr.html.

[21] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 172 – 176, 2002.