Timing-Driven Placement for FPGAs

Alexander (Sandy) Marquardt, Vaughn Betz, and Jonathan Rose[1]
{arm, vaughn, jayar}@rtrack.com

Right Track CAD Corp.,
720 Spadina Ave., Suite #313
Toronto, ON, M5S 2T9

Dept. of Electrical and Computer Engineering,
University of Toronto, 10 King's College Road
Toronto, ON, M5S 3G4

Abstract

In this paper we introduce a new Simulated Annealing-based timing-driven placement algorithm for FPGAs. This paper has three main contributions. First, our algorithm employs a novel method of determining source-sink connection delays during placement. Second, we introduce a new cost function that trades off between wire-use and critical path delay, resulting in significant reductions in critical path delay without significant increases in wire-use. Finally, we combine connection-based and path-based timing-analysis to obtain an algorithm that has the low time-complexity of connection-based timing-driven placement, while obtaining the quality of path-based timing-driven placement.

A comparison of our new algorithm to a well known non-timing-driven placement algorithm demonstrates that our algorithm is able to increase the post-place-and-route speed (using a full path-based timing-driven router and a realistic routing architecture) of 20 MCNC benchmark circuits by an average of 42%, while only increasing the minimum wiring requirements by an average of 5%.

1. Introduction

In this paper we introduce a new timing-driven placement algorithm based on the existing placement algorithm within VPR [1][2]. VPR incorporates a timing-driven router, but does not consider timing during placement. A timing-driven router can only produce routings that are as good as the placement on which the routing is performed, so to extract more speed out of an FPGA it is essential that timing-driven placement algorithms be used.

To be useful, a timing-driven placement algorithm must produce high quality placements in reasonable amounts of time without making large sacrifices in routability. Accordingly, we have developed a Simulated Annealing based timing-driven placement algorithm that satisfies all of these goals.

Our new timing-driven algorithm includes unique features that have not been previously used. First, we have developed a novel method to automatically compute and store delays between any set of locations in an island style FPGA with any routing architecture. These delays are then used during placement to efficiently evaluate source to sink connection[1] delays. Second, we present a new cost function that accurately balances wire-usage and circuit speed in any ratio. Third, we combine path-based and connection-based approaches to timing-driven placement, and we achieve an algorithm with the time complexity of a connection-based approach while achieving the quality of a path-based approach.

This paper is organized as follows. Section 2 discusses placement and timing-driven placement. Section 3 describes our new timing-driven placement algorithm, which we call T-VPlace. Section 4 describes the method by which we evaluate the quality of our algorithm. Section 5 shows how we select values for various parameters within our algorithm, while Section 6 discusses the time complexity of our algorithm. Section 7 compares the post-routing result quality of placements produced by our new algorithm to that of placements produced by a high quality routability-only placement algorithm. Finally in Section 8 we present our conclusions.

2. Background

Placement is the process by which a netlist of circuit blocks (which are either I/Os or logic blocks) are mapped onto physical locations in an FPGA. In this section we first discuss Simulated Annealing, which is an algorithm that is commonly applied to placement problems. Then we present the VPR placement algorithm, which we have enhanced to

1. This work was performed at the University of Toronto. Sandy Marquardt and Vaughn Betz are now with Right Track CAD, and Jonathan Rose is now at both Right Track CAD and the University of Toronto.

1. In a graph representation of a circuit we define a "connection" to be an edge between a net driver and any of its terminals.

produce our new timing-driven algorithm. After this, we discuss timing-analysis. Finally, we present some existing timing-driven placement algorithms.

2.1. Simulated Annealing

Both our new placement algorithm and VPR's original placement algorithm are Simulated Annealing based. In this section we give a brief introduction to Simulated Annealing, and discuss how it is applied to the placement problem.

The Simulated Annealing algorithm mimics the annealing process used to gradually cool molten metal to produce high-quality metal structures [12]. A Simulated Annealing-based placer initially places logic blocks and I/Os (circuit blocks) randomly into physical locations in an FPGA. Then the placement is iteratively improved by randomly swapping blocks and evaluating the "goodness" of each swap with a cost function[1]. If the move will result in a reduction in the placement cost, then the move is accepted. If the move would cause an increase in the placement cost, then the move sill has some chance of being accepted even though it makes the placement worse. The purpose of accepting some "bad" moves is to prevent the Simulated Annealing-based placer from becoming trapped in a local minimum.

2.2. The VPR Placement Tool (VPlace)

In this paper we will refer to the placement algorithm used within VPR as VPlace. VPlace is a Simulated Annealing-based placement algorithm that attempts to minimize the amount of interconnect required to route a circuit by placing circuit blocks that are on the same net close together. To accomplish this, VPlace uses a bounding-box based [1][2] cost function to estimate wire-length requirements.

The cost function used in VPlace has the following functional form [1][2]

$$Wiring\_Cost = \sum_{i=1}^{N_{nets}} q(i) \cdot [bb_x(i) + bb_y(i)] \qquad (1)$$

where there are N_{nets} in the circuit. The cost of each net, i, is determined by its horizontal span, $bb_x(i)$, and its vertical span, $bb_y(i)$. The horizontal and vertical spans of a net are demonstrated in Figure 1. The q(i) factor compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals. The values used for q(i) were obtained from

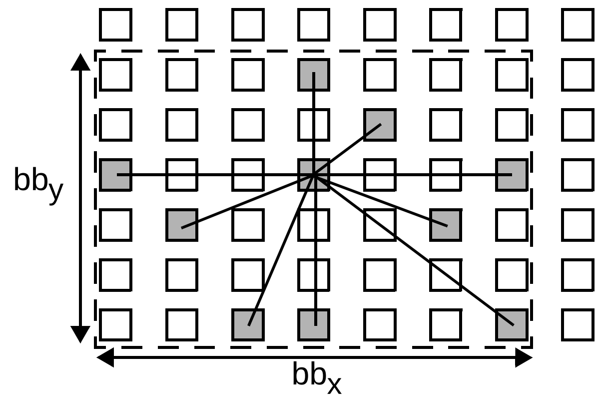

Figure 1: Example Bounding-Box of a 10 Terminal Net [1][2]

[13] so that q(i) is set to 1 for nets with 3 or fewer terminals, and it slowly increases to 2.79 for nets with 50 terminals. Beyond 50 terminals, the q(i) function linearly increases at the rate of

$$q(i) = 2.79 + 0.02616 \cdot (Num\_Terminals - 50). \qquad (2)$$

The complexity of this algorithm is $O(n^{4/3})$ where n is the number of blocks in the circuit.

2.3. An Introduction to Timing Analysis

Timing-analysis [3] must be done during timing-driven placement to compute the delay of all of the paths in a circuit. These delay values are then be used to guide the algorithm so that it can reduce the critical paths. This section describes the method that we use for timing-analyzing circuits.

We must first represent the circuit under consideration as a graph. Nodes in the graph represent input and output pins of circuit elements such as LUTs, registers, and I/O pads. Connections between these nodes are modeled with edges in the graph. These edges are annotated with a delay corresponding to the physical delay between the nodes.

To determine the delay of the circuit, a breadth first traversal is performed on the graph starting at sources (input pads, and register outputs). Then we compute the *arrival time*, $T_{arrival}$, at all nodes in the circuit with the following equation

$$T_{arrival}(i) = Max_{\forall j \in fanin(i)}\{T_{arrival}(j) + delay(j, i)\} \qquad (3)$$

Where node i is the node currently being computed, and delay(j,i) is the delay value of the edge joining node j to node i. The delay of the circuit is then the maximum arrival time, D_{max}, of all nodes in the circuit.

1. A typical cost function used in an FPGA placement algorithm may attempt to minimize the amount of wiring and/or the delay of the resulting circuits.

To guide a placement or routing algorithm, it is useful to know how much delay may be added to a connection before the path that the connection is on becomes critical. The amount of delay that may be added to a connection before it becomes critical is called the *slack* [3] of that connection. To compute the slack of a connection, we must compute the *required arrival time*, $T_{required}$, at every node in the circuit. We first set the $T_{required}$ at all sinks (output pads and register inputs) to be D_{max}. Required arrival time is then propagated backwards starting from the sinks with the following equation

$$T_{required}(i) = Min_{\forall j \in fanout(i)}\{T_{required}(j) - delay(i,j)\} \quad (4)$$

Finally, the slack[1] of a connection (i,j) driving node, j, is defined as:

$$Slack(i,j) = T_{required}(j) - T_{arrival}(i) - delay(i,j) \quad (5)$$

2.4. *Timing-Driven Placement*

Timing-driven placement algorithms attempt to map circuit blocks (IOs, logic blocks) that are on the critical path into physical locations that are close together so as to minimize the amount of interconnect that the critical signals must traverse. In this section we discuss some existing timing-driven placement tools, starting with two FPGA-specific placement tools, PROXI [4] and Triptych [5], followed by two standard cell placement tools, Timberwolf [6], and SPEED [7].

The PROXI [4] algorithm is a Simulated Annealing based timing-driven placement algorithm that is designed for FPGAs. This algorithm has good results but it is very computationally intensive. It performs simultaneous placement and routing by ripping up and rerouting all the disturbed nets after each placement perturbation during the anneal. The algorithm achieves an 8% - 15% improvement in delay compared to the Xilinx XACT 5.0 place and route system; however it has a significant disadvantage in CPU compile time, ranging from 6 times for the smallest design (a 12 x 12 array) to 11 times for the largest design (a 16 x 16 array). The computational complexity of this algorithm (while not given in [4]) appears to be $O(n^3)$ from the data they present, making it infeasible for large circuits.

The Triptych [5] CAD tools incorporate a timing-driven placement algorithm that, like ours, is based on Simulated Annealing. However, its approach uses only a single, unit-delay model timing-analysis before placement begins to evaluate the criticality of each connection, so their estimate of where the critical path lies can be inaccurate. Also, its cost function weights each source-sink connection by its criticality, and focuses on minimizing the sum of these weighted connection lengths. Hence this cost function implicitly assumes that the delay of a connection is a linear function of the Manhattan distance between its terminals. Unfortunately, this assumption is not true for most FPGA architectures — the delay of a connection is more complex.

Timberwolf [6] is Simulated Annealing based and is designed for row-based standard cell ICs. Timberwolf does a good job in reducing circuit delay, but it appears to be very computationally expensive since it is fully path based (however the computational complexity of the algorithm is not given in [6]). Additionally, the delay modeling used in Timberwolf is not realistic for circuits implemented in deep-submicron processes or for FPGAs since its delay formulation ignores the wiring resistance of individual source-sink connections. Also, Timberwolf assumes that all source-sink connections on a single net have the same delay. Using this delay formulation, Timberwolf reduces circuit delay by 28% - 50% at an area cost of 2.5% - 6% on three MCNC circuits for which timing results were available.

Another timing-driven placement algorithm is SPEED [7]. This algorithm uses quadratic programming placement techniques to place Standard Cell designs. SPEED uses a star model and the Elmore delay to compute the delay between connections on a net. First the algorithm computes a star node which is the center of gravity of all pins on the net, and then an RC tree is constructed assuming that all connections go from the driver to the star node and from the star node to each sink. This method of modeling connectivity does not accurately reflect how connections are made in an FPGA, and therefore does not accurately reflect the delay of an FPGA. Because SPEED is partitioning based and is not designed for FPGAs, it is difficult for us to say how well it could be adapted to FPGAs.

3. A New Timing-Driven Placement Algorithm: T-VPlace

We have developed a new placement tool called T-VPlace which is an extension to the original VPlace algorithm. T-VPlace is both wireability-driven (minimizing wiring requirements) *and* timing-driven. It is essential to consider both the goal of minimizing wiring and reducing critical path delay because a timing-driven only approach will lead to circuits that require an unacceptable amount of routing resources, while considering only wireability will lead to slow circuit implementations. T-VPlace simultaneously considers critical path delay and wireability and finds a reasonable compromise between the two. T-VPlace is simulated annealing-based and it uses the same annealing

1. Slack is the amount of delay that can be added to the connection before causing any path that it is on to become critical.

```
S = RandomPlacement ();
T = InitialTemperature ();
Rlimit = InitialRlimit ();
Criticality_Exponent = ComputeNewExponent();

ComputeDelayMatrix();

while (ExitCriterion () == False) {          /* "Outer loop" */

    TimingAnalyze();            /* Perform a timing-analysis and update each connections criticality */
    Previous_Wiring_Cost =Wiring_Cost(S);     /* wire-length minimization normalization term */
    Previous_Timing_Cost = Timing_Cost(S);                 /* delay minimization normalization term */

    while (InnerLoopCriterion () == False) {     /*  "Inner loop" */

        Snew = GenerateViaMove (S, Rlimit);
        ΔTiming_Cost = Timing_Cost(Snew) - Timing_Cost(S);
        ΔWiring_Cost =Wiring_Cost(Snew) -Wiring_Cost(S);
        ΔC =  λ·(ΔTiming_Cost/Prev_Timing_Cost) +
                                             (1-λ)·(ΔWiring_Cost/Previous_Wiring_Cost); /* new cost fcn */
        if (ΔC < 0) {
            S = Snew  /* Move is good, accept */
        }
        else {
            r = random (0,1);
            if (r < e^(-ΔC/T)) {
                S = Snew;  /* Move is bad, accept anyway */
            }
        }
    }            /* End "inner loop" */

    T = UpdateTemp ();
    Rlimit = UpdateRlimit ();
    Criticality_Exponent = ComputeNewExponent();

}    /* End "outer loop" */
```

Figure 2: Pseudo-code for T-VPlace.

schedule as the original VPlace algorithm. Figure 2 shows the pseudo-code for the T-VPlace algorithm.

The following sections describe the T-VPlace algorithm in detail, including descriptions of our novel method of modeling delay, our new cost function, and a brief description of our approach to timing analysis.

3.1. Delay Modeling

To maximize the performance and quality of T-VPlace, we must accurately and efficiently model the delay of each connection in the circuit as the circuit is placed. The most accurate technique would be to route each proposed placement and extract the routed delay of each connection as [4] did, but this approach requires unacceptable amounts of CPU time. Instead, we create a "delay profile" of an FPGA that is used to rapidly evaluate the delay of each connection in the circuit given the placement of its terminals.

In a "tile-based" FPGA, the FPGA structure is homogeneous, i.e. every x,y location in the FPGA is constructed from identical tiles. All current island style FPGAs are composed of one identical tile, or a small number of distinct but nearly identical tiles. We exploit the uniformity of such architectures by computing the delay of a connection between two blocks as a function only of the distance (Δx, Δy) between them. To allow an efficient assessment of the delay between blocks that are Δx and Δy distance apart, we compute a *delay lookup matrix* indexed by Δx and Δy. To compute a given (Δx, Δy) entry in the matrix, we employ the VPR router to determine the delay between two blocks that are (Δx, Δy) distance apart. To do this, a source block is placed at a location (x_{source}, y_{source}) in the FPGA, and a sink block is placed at ($x_{source}+\Delta x$, $y_{source}+\Delta y$). Then VPR's timing-driven router is used to perform a routing between the two blocks, and the delay is recorded in the delay lookup matrix at location (Δx, Δy). This process is then repeated for every possible Δx and Δy value in the FPGA. Notice that our connection delay estimate does not depend on a net's fanout — the timing-driven router we use (VPR's timing-driven router) automatically inserts buffers during routing, so the delay of time critical connections is not strongly dependent on fanout.

Since we use the timing-driven router to compute the delay between blocks, we are able to take advantage of all of the architectural features in the FPGA. For example if two blocks are on opposite sides of the FPGA and there is a long line crossing the FPGA, the timing-driven router will recognize this and the delay lookup matrix will reflect the smallest possible delay (the one using the long line) between the two locations. The reason that we use the smallest possible delay between two blocks to compute the values in the delay lookup matrix is because we know that after placement, the router will be smart enough to use the fastest resource to connect two locations on the critical path(s). By using the timing-driven router to profile the delay as a function of distance in this way, we ensure that T-VPlace automatically adapts to different FPGA architectures.

3.2. Cost Function

We need a cost function that reduces delay of connections on the critical path, and allows the delay of non-critical connections to be increased. To properly balance the trade-off between wire-length minimization and critical path minimization, we have developed a new cost function that we call the *auto-normalizing* cost function. Before we discuss this new cost function we need to introduce some definitions.

We first introduce a new term called *Timing_Cost* for each source sink pair, (i, j). This is the portion of the cost function that will be responsible for minimizing the critical path delay. Timing_Cost is based on the *Criticality* of each connection, the *Delay* of each connection, and a user defined *Criticality_Exponent*. The Delay for each connection is obtained from the delay lookup matrix and the current placement, the Criticality_Exponent is defined below, and Criticality[1] is defined as follows

$$Criticality(i, j) = 1 - \frac{Slack(i, j)}{Dmax} \qquad (6)$$

where D_{max} is the critical path delay (maximum arrival time of *all* sinks in the circuit), and Slack is the amount of delay that can be added to a connection without increasing the critical path delay.

1. Note that this equation assumes single clock circuits.

In our new cost equation, to control the relative importance of connections with different criticalities, we compute a power of the Criticality of each connection depending on a variable called Criticality_Exponent (i.e. $Criticality^{Criticality\_Exponent}$). The purpose of including an exponent on the Criticality in our new cost function is to heavily weight connections that are critical, while giving less weight to connections that are non-critical.

We now define the Timing_Cost of a connection, (i,j) as follows

$$Timing\_Cost(i, j) = Delay(i, j) \cdot Criticality(i, j)^{Criticality\_Exponent} \qquad (7)$$

And the total Timing_Cost for a circuit is the sum of the Timing_Cost of all of its connections as follows

$$Timing\_Cost = \sum_{\forall i, j \subset circuit} Timing\_Cost(i, j) \qquad (8)$$

We now present our auto-normalizing cost function (wiring_cost is defined in equation (1)):

$$\Delta C = \lambda \cdot \frac{\Delta Timing\_Cost}{Previous\_Timing\_Cost} + (1 - \lambda) \cdot \frac{\Delta Wiring\_Cost}{Previous\_Wiring\_Cost} \qquad (9)$$

Our auto-normalizing cost function depends on the *change* in Timing_Cost and Wiring_Cost. It uses a trade-off variable called λ to determine how much weight to give each component. To normalize the weight of these two components we use two normalization variables called Previous_Timing_Cost and the Previous_Wiring_Cost that are updated once every temperature. The effect of these two normalization components is to make the function weight the two components only with the λ variable, independent of their actual values. This is convenient because it automatically adjusts the weights of the two components so that the algorithm is always allocating λ importance to changes in the Timing_Cost, and (1-λ) importance to changes in the Wiring_Cost.

If λ is 1 then we have an algorithm that focuses only on timing, but ignores wire-length minimization. If λ is 0, then we have the original VPlace algorithm that focuses only on minimizing wire-length. For example, if we have a λ value of 0.7, we want every move to be 70% due to changes in Timing_Cost, and 30% due to changes in Wiring_Cost. If we did not normalize, and we had Timing_Cost values that were orders of magnitude less than Wiring_Cost then the cost function would only be affected by changes in the Wiring_Cost even though we desired this to only account for 30% of the change in total cost. Another benefit of this auto-normalizing approach is that as the temperature changes, we are constantly re-normalizing the weights of the two components. Compare this to other approaches that only normalize the components once at the beginning of the algorithm [14], which means that if the two components change at different rates, this normalization will become increasingly inaccurate, and will inadvertently allocate more weight to one of the components than was desired.

We use this cost function in our algorithm without modifying the annealing schedule from VPR. Since the annealing schedule is "adaptive"[1], it performs well with our new cost function.

3.3. A New Approach to Timing Analysis

There are different approaches to minimizing critical path delay in timing-driven placement algorithms. A path-based approach to timing-driven placement involves using timing-analysis to compute path delays at every stage of the placement and using these delays in a cost function. This path-based approach is computationally expensive since moving any connection requires that all paths that go through that connection be re-analyzed. Another approach is connection-based timing-driven placement, which involves performing a path-based timing-analysis before placement, and then assigning slacks to each connection in the circuit. Then during placement more attention is paid to connections with low slack (higher criticality), but the more global view of the complete path delay is not used.

Our approach is to combine path-based timing-analysis and connection-based timing-analysis. This is accomplished by periodically performing a path-based timing-analysis after a certain number of simulated-annealing moves are completed. While the delay values used in the placement are always up to date (using the method described in Section 3.1), the slack values (obtained by this "infrequent" timing-analysis) may be based on delay values that do not precisely reflect the connection delays of the current placement. This method ensures that the computation time spent on timing-analysis does not significantly degrade the total placement compile time. We have experimentally determined how often a path-based timing-analysis must be done to get the best results, and we discuss these experiments in Section 5.

1. For a full description of the adaptive annealing schedule, see [1][2]

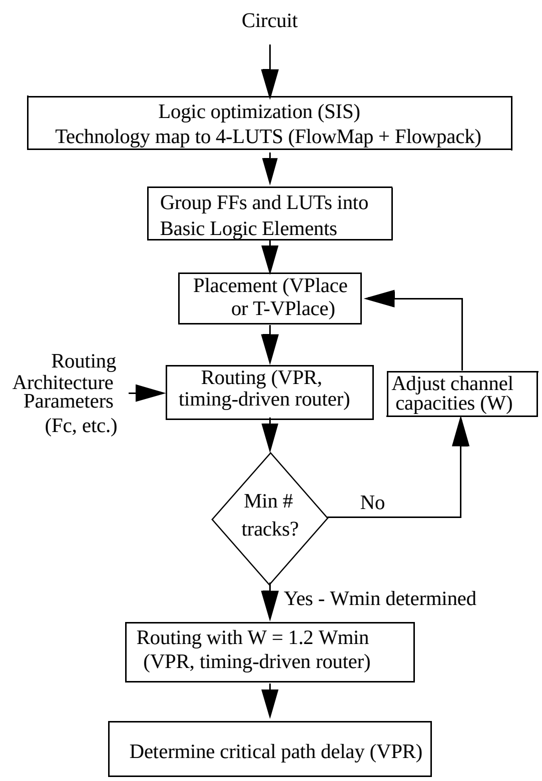

Figure 3: Architecture evaluation CAD flow [1][2].

4. Algorithm Evaluation

We use an empirical method to evaluate our placement algorithm and to compare it to VPR's original placement algorithm. This involves technology-mapping, packing, placing, and routing benchmark circuits into realistic FPGA architectures. The delay and wiring requirements of each circuit implementation is then computed using sophisticated models, and from this we are able to compare the results of our algorithm to the results of the existing VPR placement algorithm.

4.1. CAD Flow

The CAD flow that we use to evaluate different placement algorithms is basically the same as in [1][2], and is given in Figure 3. First each circuit is logic-optimized by SIS [9] and technology mapped into 4-LUTs by FlowMap [10]. Then VPack is used to group the LUTs and flip-flops into Basic Logic Elements. After this, each circuit is mapped onto an FPGA with T-VPlace. Finally, VPR's timing-driven router is used to connect all of the wiring. Note that this flow is completely timing-driven.

Figure 3 shows how VPR computes the minimum number of tracks that are required for a circuit to successfully route. Basically VPR repeatedly routes each circuit with different channel widths (number of tracks per channel), scaling the FPGA's architecture until it finds the minimum number of tracks in which the circuit will route. Note that at this minimum track count the circuit is just barely routable, so we call this a high-stress routing.

We define a *low-stress* routing to occur when an FPGA has 20% more routing resources than the minimum required to route a given circuit. We feel that low-stress routings are indicative of how an FPGA will generally be used (it is rare that a user will utilize 100% of all routing and logic resources) so this is the utilization that we use to evaluate the speed of each circuit. We also evaluate how well the router could do if there were an infinite amount of routing resources in the FPGA. The critical path delay obtained from these *infinite-resource* routings indicates the circuit speed achievable in a very routing-rich FPGA, which we feel is a good initial indicator of how well the placement algorithm has performed. For our post-place-and-route experiments we present both low-stress and infinite-resource critical path delay numbers.

4.2. FPGA Architecture

The routing architecture that we use to evaluate our new placement algorithm is an island style architecture similar to the Xilinx Virtex [11] part. All the wires in the routing architecture we use are of length 4 — i.e., each wire spans four logic blocks before terminating. Half of the programmable switches in the FPGA routing are tri-state buffers, while the other half are pass transistors. Previous research has shown that this routing architecture has good performance [1][2]. Also, we assume that the FPGA is composed of logic blocks containing a single 4-LUT and a flip-flop. We use this logic block for our experiments because we wish to compare placement algorithms on large circuits, and using small logic blocks effectively makes the benchmark circuits larger (there are more blocks to place). We believe that it is beneficial to perform experiments on large circuits, typical of the designs being implemented in high-capacity FPGAs today, in order to obtain the most accurate conclusions about our algorithms.

5. Algorithm Tuning

In our algorithm we tune various parameters to get the best performance. We must find the best value for λ (which determines the trade-off between wire use and critical path delay), the best Criticality_Exponent, and we must determine how often to timing analyze the circuits as the placement evolves. To find the best values for these parameters, we performed experiments on the 20 largest MCNC circuits with the routing architecture described in Section 4.2.

By using the delays from the delay lookup matrix annotated onto connections in the circuits, we are able to obtain critical path delay estimates from the placement algorithm without performing a routing. These estimates allow us to fairly compare the performance of T-VPlace with different algorithm parameters in a reasonable amount of computation time. We will later show in Section 5.4 that these placement estimates of the critical path are a good tool (have good fidelity with respect to the final routed delay) for comparing algorithm performance.

5.1. Timing-Analysis Interval

The first parameter that we discuss is the timing-analysis interval. For this experiment we set the value of λ to 1 (fully timing-driven) and the Criticality_Exponent to 1. We then vary how often we timing-analyze the circuit and update the connection criticalities and slacks. The sweep goes from once at the beginning of execution all the way up to timing-analyzing within the inner loop of the placement algorithm (see Figure 4 for the pseudo-code of the algorithm). We present two tables showing the effect of this timing-analysis interval. The first results shown in Table 1.1 are for timing analysis performed in the outer loop of the placement algorithm. This first column in this table shows the number of temperature changes between each timing-analysis (which we call the timing-analysis interval), the second column shows the placement estimated critical path, and the third column shows the Wiring_Cost.

TABLE 1.1 Effect of timing-analysis in the outer loop

Timing-Analysis Interval	**Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)**	**Wiring Cost (20 Circuit Geometric Average)**
1	39.3	529.6
2	39.5	531.1
4	40.1	530.5
8	40.5	531.0
16	39.5	530.3
32	41.4	534.5
64	41.3	528.3
128	43.0	522.9
Never	43.0	522.9

Table 1.2 shows the effect of timing-analyzing the circuit in the inner loop of the placement algorithm. The first column shows how many times timing-analysis is performed in the

inner loop of the annealer at each temperature; the second column shows the placement estimated critical path; the third column shows the Wiring_Cost.

TABLE 1.2 Effect of timing-analysis in the inner loop

Number of Timing-Analysis in the Inner Loop at Each Temperature	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	Wiring Cost (20 Circuit Geometric Average)
1	39.3	529.6
10	39.2	528.8
50	40.1	525.6
100	39.7	530.9

These results indicate that performing a timing-analysis once per temperature is sufficient to obtain the best placement results. It appears that this re-analysis interval does not affect the bounding box (wirelength) cost of the placement. For the remainder of our experiments, we set T-VPlace to timing-analyze each circuit once per temperature change.

5.2. Criticality Exponent

The next parameter that we will discuss is the Criticality_Exponent. We have performed two sets of experiments to determine the best value for the Criticality_Exponent. In the first experiment we have set λ to 0.5. We then performed the same experiments with λ set to 1. Again, all of the results presented are the placement estimated critical paths and Wiring_Cost.

We first show the effect of the different Criticality_Exponents when $\lambda = 0.5$ in Table 1.3. These results show that increasing the criticality exponent up to about 8 or 9 improves the placement estimated critical path, at which point no more gains are apparent. These results also show that large exponents improve the Wiring_Cost. This fact deserves more discussion.

Large exponents make very few connections have a very large Timing_Cost, and all other connections have an insignificant Timing_Cost. Because we normalize Timing_Cost and Wiring_Cost, we ensure that no matter how large the Timing_Cost of a connection becomes, it will still only account for a fixed percentage of the total normalized cost. This means that a high Criticality_Exponent results in fewer connections in the circuit being critical, but for these few connections the Timing_Cost makes up the largest portion of the normalized cost. For other non-critical connections (which we know there are more of as the Criticality_Exponent is increased), the Wiring_Cost makes up the largest portion of the normalized cost. As a result, the placement algorithm is able to focus on minimizing wiring requirements for more nets as the Criticality_Exponent is increased.

TABLE 1.3 Effect of Criticality_Exponent with a λ value of 0.5.

Criticality Exponent	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	Wiring Cost (20 Circuit Geometric Average)
1	38.9	342.0
2	37.1	343.4
3	35.9	344.0
4	34.8	344.7
5	34.7	343.7
6	34.8	341.6
7	34.3	339.6
8	34.3	340.1
9	33.8	339.6
10	34.3	337.9
11	34.3	336.3

TABLE 1.4 Effect of Criticality_Exponent with a λ value of 1

Criticality Exponent	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	Wiring Cost
1	39.3	529.6
2	36.4	540.9
3	36.1	567.4
4	37.6	593.3
5	36.5	623.8
6	40.2	681.0
7	43.8	717.3

The next experiment (Table 1.4) shows that when $\lambda = 1$ (meaning the cost is purely timing based), an exponent value of 2 or 3 is the best. Compared to the results that we displayed in Table 1.3 the critical path is worse, and the Wiring_Cost is much worse. It is surprising that the delay results for a λ value of 1 are worse than a λ value of 0.5 since in $\lambda = 1$ case, the algorithm is only attempting to minimize delay, while in the $\lambda = 0.5$ case the algorithm is considering both delay and wire-length minimization. This result deserves more discussion.

When we set up the algorithm to only minimize delay (by setting λ=1), it attempts to minimize the current critical path at the cost of extending other non-critical paths. Since we are only timing-analyzing the circuit once per temperature, the algorithm has many moves between updates of the connection criticalities and slacks. This means that it is likely that the algorithm is able to significantly reduce critical paths during one iteration of the outer loop, but at the same time inadvertently make other paths very critical. This oscillation effect makes it difficult for the placement algorithm to converge to the best placement solution.

By including a wire-length minimization term in the cost equation, we are able to reduce the oscillations of the placement. This is because the wire-length term will penalize moves that significantly increase the wire-length of the placement, making them unlikely to be accepted even if they would significantly reduce the current critical path. Effectively, the wire-length term acts as a damper on the delay minimization term in our cost function, and prevents oscillation.

5.3. *Trade-off Between Wireability and Critical Path Delay*

Now we are ready to evaluate the effect of the trade-off parameter λ. The above results show that using a timing-analysis interval of once per temperature, a Criticality_Exponent value of 8, and λ of 0.5 provides the best results so far. Based on these results, we set the timing analysis interval to once per temperature, the Criticality_Exponent to 8, and we varied λ. The results of this experiment are shown in Table 1.5.

TABLE 1.5 Effect of λ with a Criticality_Exponent of 8 and timing-analysis interval of 1.

λ	**Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)**	**Wiring Cost (20 Circuit Geometric Average)**
0	51.6	312.7
0.1	40.0	315.8
0.2	37.8	318.5
0.3	36.7	322.8
0.4	35.6	331.1
0.5	34.0	339.8
0.6	33.2	353.6
0.7	32.5	373.9
0.8	32.5	400.7
0.9	32.4	439.7
1	43.4	725.3

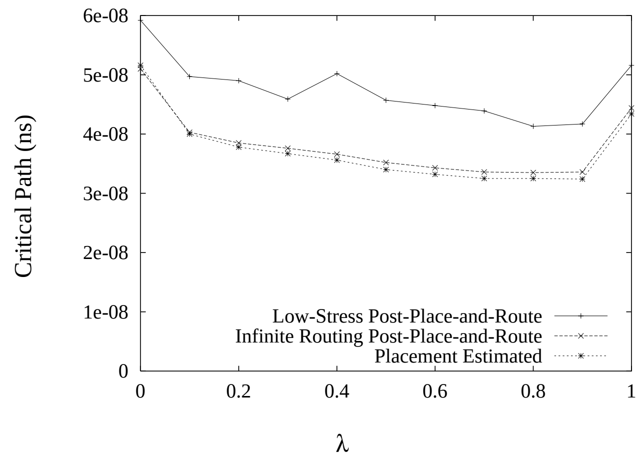

Figure 4: Graph showing the fidelity of the placement estimated critical path.

This table shows that an algorithm that is only wire-length driven produces the best Wiring_Cost. It also shows that an algorithm with a λ of 0.9 produces circuits with the best placement estimated critical path delay. A λ of 1 is bad for both critical path, delay, and wire-length for the reasons explained above. We feel that setting λ to 0.5 provides the best compromise between wire-length and critical path minimization, so the remainder of our experiments use this value.

5.4. *Verification of the Fidelity of the Placement Estimated Critical Path Delay*

In the previous section we used placement-estimated critical path delays to tune the parameters used in the placement algorithm. It is interesting to see how well this estimate correlates to the actual post-place-and-route critical path delays. To study the correlation, we present a λ sweep graph with a Criticality_Exponent of 8 in Figure 4. This graph shows the infinite routing resource post-place-and-route delay, the low-stress post-routing delay, and the placement estimated post-place-and-route delay. There is an excellent correlation between the placement estimated critical path and the infinite routing-resource critical path. Additionally the low-stress results follow the same trend as the placement-estimated results. We therefore believe that it is valid to use the placement-estimated delay results as an evaluation metric as we did in the previous section.

6. Complexity Analysis

The complexity of our algorithm is essentially the same as VPlace. We perform a timing analysis once per temperature change which is an O(n) operation. At each temperature we execute the inner loop of the placer $O(n^{4/3})$ times (i.e. we perform $O(n^{4/3})$ swaps). In the inner loop we have an incremental-bounding-box-update operation that is worst case $O(k_{max})$, where k_{max} is the fanout of the largest net in the circuit. The average case complexity for this bounding box update is O(1) [1][2]. Also in the inner loop is the computation of the Timing_Cost for each connection affected by a swap. This is also $O(k_{max})$. In the average case this is $O(k_{avg})$ where k_{avg} is the average fanout of all nets in the circuit. Since k_{avg} is typically quite small, the average complexity of this Timing_Cost computation is O(1) as well. The overall result is that our algorithm is worst case $O[(k_{max} \cdot n)^{4/3}]$, but on average it is $O(n^{4/3})$[1]. T-VPlace takes about 2.25 times as long as VPlace to place the largest MCNC circuit (clma, which consists of 8300 LUTs) — about 9 minutes vs. 4 minutes on a 450 MHz Pentium.

1. The average case complexity is really the only relevant value here. The complexity of the algorithm is the average over millions of swaps, so a user will always see the average case complexity.

TABLE 1.6 Post-place-and-route comparison of VPlace and T-VPlace (cluster size = 1).

Circuit	Post-Place-and-Route Minimum Channel Width (W_{min})			Post-Place-and-Route Critical Path (ns) $W = \infty$			Post-Place-and-Route Critical Path (ns) $W = W_{min} + 20\%$		
	VPlace	**T-VPlace ($\lambda = 0$)**	**T-VPlace ($\lambda = 0.5$)**	**VPlace**	**T-VPlace ($\lambda = 0$)**	**T-VPlace ($\lambda = 0.5$)**	**VPlace**	**T-VPlace ($\lambda = 0$)**	**T-VPlace ($\lambda = 0.5$)**
alu4	14	14	14	40.3	40.4	29.8	42.4	41.2	33.4
apex2	15	17	16	46.9	46.3	32.3	47.7	46.5	48.8
apex4	17	16	18	40.9	44.8	28.2	42.0	46.8	31.7
bigkey	13	13	10	36.0	35.2	21.6	36.7	35.4	25.2
clma	16	16	17	90.2	91.1	72.3	116.0	166.0	130.0
des	11	12	11	40.5	48.9	30.2	50.4	57.4	43.7
diffeq	11	11	12	35.2	37.5	30.8	38.9	41.0	34.9
dsip	12	12	12	27.9	27.2	21.7	28.3	28.8	22.9
elliptic	14	16	15	70.6	76.1	46.1	79.5	79.6	58.1
ex1010	14	15	15	85.0	77.5	52.9	96.2	78.6	70.5
ex5p	17	17	19	39.6	40.4	28.1	42.7	42.7	43.5
frisc	16	17	18	70.8	73.2	59.6	76.8	79.6	61.6
misex3	14	15	15	39.0	40.2	26.6	39.3	75.0	34.3
pdc	22	21	24	81.7	74.5	49.9	122.0	114.0	73.0
s298	11	12	12	74.8	72.0	53.6	116.0	78.7	77.8
s38417	11	11	12	61.7	71.0	33.7	70.0	74.6	37.2
s38584.1	11	11	11	45.3	44.1	31.8	49.7	44.3	36.4
seq	16	16	16	45.7	41.0	28.1	46.4	43.7	39.5
spla	18	18	20	58.4	67.4	39.7	74.8	100.0	69.4
tseng	9	10	11	33.7	33.1	28.3	39.8	38.4	33.1
Geom. Av.	13.78	14.22	14.50	50.1	51.0	35.2	57.1	59.2	45.7
%diff w.r.t VPlace	—	+3.2%	+5.2%	—	+1.8%	-29.7%	—	+1.04%	-20.0%

7. Results: VPlace vs. T-VPlace

In this section we compare the post-place-and-route results from VPlace and T-VPlace. Again, our results are obtained by implementing 20 MCNC benchmark circuits in the FPGA architecture described in Section 4.2. Additionally, all of the results that we present are based on a Criticality_Exponent of 8, and a timing-analysis interval of once per temperature change.

The results we show are post-place-and-route for both VPlace and T-VPlace. Table 1.6 shows that for the infinite routing case, T-VPlace improves circuit speed by about 42% (a 30% decrease in delay) on average compared to VPlace. For the low stress routing case, T-VPlace improves circuit speed by 25% (a 20% reduction in delay) on average compared to VPlace. The cost of this speed gain is only a 5% increase in the minimum channel width. It is likely that the low-stress routing results do not show the same improvement in speed as the infinite routing results due to the fact that the placement algorithm has made it more difficult for the router to optimize the critical path(s). This is because T-VPlace produces circuits that have shorter critical paths than VPlace, but more of them. The result is that the router has many more paths to shorten, making it more difficult in the low-stress routing case for the router to get close to the "lower bound" that the infinite routing results represent.

8. Conclusions

In this paper we discussed our new timing-driven placement algorithm, T-VPlace. This algorithm has several new features of interest. In particular, it performs a delay profiling of an FPGA to allow architecture-independent and CPU efficient timing-driven placement. It also uses an auto-normalizing cost function that allows the user to specify any desired trade-off between delay and wirelength throughout the entire placement anneal. We also introduced a new combination of connection-based and path-based approaches to timing-analysis. Finally, we experimentally determined good values for various cost parameters, and showed how these values impact both the wireability and delay of circuit placements.

We showed that T-VPlace is both CPU-efficient, requiring only 2.5x more CPU time than a high quality wirelength-driven placement algorithm, and that it significantly improves circuit speed (on average by 42%). Our new T-VPlace algorithm accomplishes this improvement at a cost of only a 5% increase in the wiring requirements relative to the completely wirelength-driven VPlace algorithm. Overall it is clear that timing-driven placement can significantly improve performance without sacrificing a large amount of area.

9. References

[1] V. Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," *Ph. D. Dissertation, University of Toronto,* 1998.

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

[3] R. Hitchcock, G. Smith and D. Cheng, "Timing Analysis of Computer-Hardware," *IBM Journal of Research and Development*, Jan. 1983, pp. 100 - 105.

[4] S. Nag and R. Rutenbar, "Performance-Driven Simultaneous Place and Route for Row-Based FPGAs", *ICCAD*, 1995, pp. 332 - 338.

[5] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on VLSI*, Vol. 3, No. 4, Dec 1995.

[6] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," *DAC*, 1995, pp. 211 - 215.

[7] B. Riess and G. Ettelt, "SPEED: Fast and Efficient Timing Driven Placement," *IEEE International Symposium on Circuits and Systems*, 1995, pp. 377 - 380.

[8] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report,* Microelectronics Center of North Carolina, 1991.

[9] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Analysis," *Tech. Report No. UCB/ERL M92/41,* University of California, Berkeley, 1992.

[10] J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD,* Jan. 1994, pp. 1-12.

[11] Xilinx Inc., "Virtex 2.5 V Field Programmable Gate Arrays", *Advance Product Data Sheet*, 1998.

[12] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, May 13, 1983, pp. 671 - 680.

[13] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *ICCAD*, 1994, pp. 690 - 695.

[14] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," *DAC*, 1995, pp. 211 - 215.

[15] A. Marquardt, "Cluster-Based Architecture, Timing-Driven Packing, and Timing-Driven Placement for FPGAs," *M.A.Sc. Thesis, University of Toronto,* 1999.