Track Placement: Orchestrating Routing Structures to Maximize Routability

Katherine Compton[1], and Scott Hauck[2]

[1] Northwestern University, Evanston, IL USA
kati@ece.northwestern.edu
[2] University of Washington, Seattle, WA USA
hauck@ee.washington.edu

Abstract. The design of a routing channel for an FPGA is a complex process requiring a careful balance of flexibility with silicon efficiency. With a growing move towards embedding FPGAs into SoC designs, and the new opportunity to automatically generate FPGA architectures, this problem is even more critical. The design of a routing channel requires determining the number of routing tracks, the length of the wires in those tracks, and the positioning of the breaks between wires on the tracks. This paper focuses on the last problem, the placement of breaks in tracks to maximize overall flexibility. Our optimal algorithm for track placement finds a best solution provided the problem meets a number of restrictions. Our relaxed algorithm is without restrictions, and finds solutions on average within 1.13% of optimal.

1 Introduction

The design of an FPGA interconnect structure has usually been a hand-tuning process. A human designer, with the aid of benchmark suites and trial-and-error, develops an interconnect structure that attempts to balance flexibility with silicon efficiency. Often, the concentration is on picking the number and length of tracks – long tracks give global communication but with high silicon and delay costs, while short wires can be very efficient only if signals go a relatively short distance.

An area that can sometimes be ignored is the placement of the breaks in these interconnect wires. If we have a symmetric interconnect, with N length-N wires, we simply break one wire at each cell. However, for more irregular interconnects, it can be difficult to determine the best positioning of these breaks.

While a manual solution may be feasible in many cases when only a single architecture is being examined, it is not always practical. For example, track placement becomes extremely critical when we consider automatic generation of FPGA architectures for systems-on-a-chip [1]. In this case, a track placement may be performed a very large number of times within the inner loop of an architecture generator, and a fast but effective algorithm for automatic track placement becomes a necessity.

In this paper, we address the issue of routing architecture design for reconfigurable architectures with segmented channel routing, such as RaPiD [2] and Garp [3]. We formalize the track placement problem, define a cost metric, and introduce track placement algorithms, including one proven optimal for a subset of these problems.

Achieving the best track placement requires a careful positioning of the breaks on multiple, different-length, routing tracks. For example, in Fig. 1 right, the breaks between wires in the routing tracks are staggered. This helps to provide similar routing options regardless of location in the array. If instead all breaks were lined up under a single unit, as in Fig. 1 left, a signal might become very difficult to route if its source was on one side of the breaks and at least one sink on the other. When large numbers of tracks or tracks of different wire lengths are involved, it can become difficult to find a solution where the breaks are evenly distributed through the array.

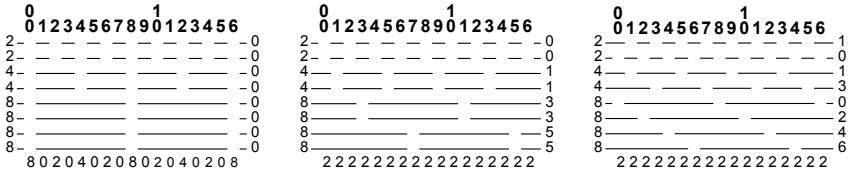

Fig. 1. Three different track placements for the same problem. A very poor one (left), an improved one (middle) and an even better placement (right). In each, the numbers on top indicate position in the architecture, and the numbers on the bottom indicate the number of breaks at the given position. Each track has its associated wire length at left, and offset at right.

The issue of determining the positioning (or offset) of a track within an architecture is referred to as track placement. The goal is to pick the offset that should be used for each track in order to maximize the routability, given a predetermined set of tracks with fixed length wires. For simplicity, each track is restricted to contain wires of only one length, which is referred to as the track's S value, or track length. The actual determination of the quantity and S values of tracks is discussed elsewhere [1], as are issues specific to 2D routing architecture design [4].

2 Problem Description

Finding a good track placement is a complex task. Intuitively, we wish to space tracks with the same S value evenly, such as by placing the length-8 tracks from the problem featured in Fig. 1 at offsets 0, 2, 4, and 6. However, a placement of tracks of one S value can affect the best placement for tracks of another S value. For example, Fig. 1 right shows that the length-4 tracks are placed at offsets 1 and 3 in order to avoid having the breaks of those tracks fall at the same locations as the breaks from the length 8 tracks. This effect is called "correlation" between the affected tracks. Correlations occur between tracks when their S values contain at least one common prime factor. It is these correlations that make track placement difficult.

From the previous example we might conclude that a possible metric for measuring the quality of a track placement would be to compute the "evenness" or "smoothness" of the break distribution throughout the architecture. However, the smoothness metric fails to capture the idea of maintaining similar routing options for every location in the array. The placement in Fig. 1 center has the same smoothness of breaks as the solution at right, but is not an equally good solution. For example, although there are two length-4 tracks, each logic unit position is at the same location along the

length-4 wires in both tracks. On the other hand, the architecture at right provides for two different locations along the length-4 wires at every logic unit position. For this reason, we consider the placement at right to be superior in terms of routing options at each position despite the two architectures having the same break smoothness.

Instead, our chosen goal in track placement is to ensure that signals of all lengths have the most possible routes. To quantitatively measure this routability, we examine each possible signal length, and find the region of the interconnect that gives the fewest possible routes for this length signal. Summing across all possible signal lengths gives yields the "diversity score" of a track placement, as shown in Equation 1. The fewer and smaller the routing bottlenecks, the more routing choices are available throughout the architecture, and the higher the diversity score.

Equation 1. The diversity score for a particular track placement routing problem T and solution set of offsets O for the tracks in T can be calculated using the equation:

$$diversity\_score(T,O) = \sum_{L}\left(\min_{all\ \ positions}\left(\sum_{Ti \in T} uncut(T_i, O_i, L, position) \right) \right)$$

for all tracks Ti with their given wire lengths and offsets Oi, possible signal lengths L, and all possible positions in the interconnect. The uncut() function returns a binary result that is 0 if there is a break within the range [position, position + L) on track Ti that has been placed at offset Oi, and 1 otherwise.

Fig. 2 shows the diversity score calculation for two different placements of the same track problem. Here we consider a four position window that encapsulates the full repeating pattern of breaks of the placement. The length of the window that needs to be considered can be found by taking the LCM of all S values in the problem. Examining a larger window would simply yield the same results. Within this window we count the number of tracks at each position that can be used to route a signal of the given length towards the right. The different possible signal lengths (L) are listed, and at each position, the number of tracks useable to route that signal length is given.

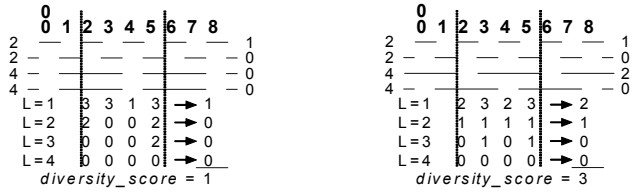

Fig. 2. Diversity score for two different track placements. Positions are given by the top row of numbers. For each track, segment length S is at left, and offset O is at right. The number of routing possibilities is given for each potential signal length L. Diversity score is at bottom.

In Fig. 2 left, two different tracks can be used to route length-2 signals to the right at position 2, but at position 3, no tracks are useable. At right, exactly one track can be used to route length-2 signals from any position in our window. We then take the minimum value at each length (representing a routing bottleneck for that signal length), and sum across L values to arrive at the diversity score.

We have also determined a bound on this value, as described in Theorem 1 (proven elsewhere [5]). Note that comparing diversity scores is only valid across different solutions to the same track placement problem. We discuss the issue of determining the actual quantity and types of tracks that should be used elsewhere [1].

Theorem 1. For all possible offset assignments O to track set T,

$$diversity\_score(T,O) \le \sum_{L} floor\left(|T| - \sum_{Ti \in T} \min(1, L/S_i) \right)$$

We focus on the worst-case (the regions with the fewest possible routes) instead of the average case. The average number of possible routes for a signal is independent of track placement, and only depends on the number of tracks and their wire lengths. Thus, an average case metric cannot tell the difference between the placements in Fig. 2, while the worst-case will shows that the rightmost version is superior.

3 Proposed Algorithms

We have developed a number of algorithms to solve the track placement problem based on the diversity score cost function. These track placement algorithms, both optimal and heuristic, are discussed in depth in the next few sections.

3.1 Brute Force Algorithm

Using a brute force approach, we can guarantee finding a solution with the highest possible diversity score. However, examining the entire solution space is a very slow process. The number of cases that the brute force approach must examine for a given problem is the product, over all distinct track lengths in the problem, of a multichoose of the track length and quantity of that length. For example, a modest architecture with 8 length-12 tracks, 4 length-6 tracks and 2 length-4 tracks will require the examination of *over 95 million* distinct cases. Since one of our targeted applications is within an inner loop of an automatic architecture generator [1], this approach is far too slow. Instead, it provides a bound on the diversity score, which is used to verify the results of our optimal algorithm and measure the quality of our other algorithms.

3.2 Simple Spread Algorithm

The Simple Spread algorithm is a simple heuristic for track placement. Tracks are grouped by segment length. For each group, the tracks are spaced evenly within the offsets 0 through S_i-1 (where S_i is the segment length of that group), regardless of the placement of tracks from other groups. This algorithm is simple in design and fast in execution, but disregards correlations between S values. It is included to demonstrate in the results section the necessity of considering these correlations.

3.3 Optimal Factor Algorithm

One of our goals was to develop a fast algorithm that, with some restrictions, would provably find the optimal solution. The Optimal Factor algorithm is the culmination

of many theorems and proofs. While many of the theorems will be presented here, due to reasons of space, their proofs are presented elsewhere [5].

Any optimal algorithm must consider the correlations between breaks both within an S group and across S groups. Correlations between two different S values occur when those S values share a common factor. For example, a track with S=6 and one with S=2 (a common factor of 2) will either have breaks at the same location once every 6 positions, or not at all, depending on the offsets chosen for the two tracks. On the other hand, a track with S=2 and one with S=5 will have breaks at the same location once every 15 positions regardless of the offsets chosen, as they are completely uncorrelated (no common factors). The following theorem is therefore used to divide a problem into smaller independent (uncorrelated) problems when possible.

Theorem 2. If T can be split into two sets G1⊂T and G2 = T-G1, where the S values of all tracks in G1 are relatively prime with the S values of all tracks in G2, then *diversity_score*(T,O) = *diversity_score*(G1, O1) + *diversity_score*(G2, O2), where O is the combination of sets O1 and O2. Thus, G1 and G2 may be solved independently.

We can also use the fact that correlation is based on common factors to further simplify the track placement problem. Theorem 3 states that whenever one track's S value has a prime factor that is not shared with any other track in the problem (or a track has a higher quantity of a prime factor than any other track in the problem), the prime factor can be removed. For example, with 2 length-6 tracks and one length-18 track, we can remove a 3 from 18, and essentially have 3 length-6 tracks, which can then be placed evenly using a later theorem. We do not actually change the S value of the length-18 track, just the effective S value used during track placement. After the offsets of tracks are determined, the original S values are restored if necessary.

Theorem 3. If the S_i of unplaced track T_i contains more of any prime factor than the S_j of each and every other track T_j ($i \neq j$), then for all solutions O, *diversity_score*(T, O) = *diversity_score*(T', O), where T' is identical to T, except T'_i has $S'_i = S_i$ with the unshared prime factor removed. This rule can be applied recursively to eliminate all unshared prime factors.

Next the tracks are grouped by S value (which may have been factored using Theorem 3). These groups are then sorted such that the largest S value is considered first. Any full sets (a set of N tracks all with Si = N) are removed from the problem by Theorem 4, placing one track from the set at each possible offset 0 to N-1. Note we are only considering as possible offsets the range 0…|Si|-1 for a track with S=Si. While it is perfectly valid to place a track at an offset greater than Si-1, the fact that all wires within the track are of length Si causes a break to be located every Si locations. Therefore, there will always be a break on track Ti within the range 0 to Si-1.

The basic idea of the remainder of the algorithm is to space the tracks of the largest S value (S_{max}) evenly using Theorems 4 and 5, then remove these tracks from further consideration. However, in order to respect the effect that their breaks have on the placement of tracks from successive S groups, we add pre-placed placeholder tracks (with S = the next largest S value S_{next}) such that they have the same number of breaks at the same positions as the original tracks, but in terms of S_{next} instead of S_{max}. This enables us to determine the best offsets for the real tracks with $S=S_{next}$. This process is repeated within a loop, where S_{next} becomes S_{max}, and we find the new S_{next}.

Theorem 4. Given a set G⊂T of tracks, each with S=|G|. If there exists a solution O' for tracks T'=T-G that meets the bound, then there is also solution O for tracks T that meets the bound (and thus is optimal), where each track in G is placed at a different offset 0…|G|-1.

We do set a number of restrictions, however, in order to ensure that at each loop iteration, (1) we can perfectly evenly space the tracks with $S=S_{max}$, and (2) the breaks from the set of tracks with $S=S_{max}$ can be exactly represented by some integer number of tracks with $S=S_{next}$. These restrictions are outlined in the next two theorems.

Theorem 5. Let S_{max} be the maximum S value currently in the track placement problem, M be the set of all tracks with $S = S_{max}$, N = |M|, and S_{next} be the largest $S \neq S_{max}$. If N>1, S_{max} is a proper multiple of N, and $S_{next} \leq S_{max}*(N-1)/N$, then any solution to T that meets the bound must evenly space out the N tracks with $S = S_{max}$. That is, for each track M_i, with a position O_i, there must be another track M_j with a break at $O_i + S_{max}/N$.

Theorem 6. Given a set of tracks G, all of segment length X, where X is evenly divisible by |G|, and a solution O with these tracks evenly distributed. There is another set of tracks G', all of length $Y = |G'| * X / |G|$, with a solution O' where the number of breaks at each position is identical for solution O of G and solution O' of G'. If solution in which G has been replaced with G' meets its bound, the solution with the original G also meets its bound.

```
Optimal_Factor(T) {
  Run independently on relatively prime track sets (Th. 2)
  Factor out unshared prime factors from S values (Th. 3)
  While tracks exist without an assigned Oi {
    Place and remove any full sets (Th. 4)
    If all tracks have their Oi assigned, end.
    Let Smax = the largest Si amongst unplaced tracks
    Let Snext = 2nd largest Si amongst unplaced tracks
    Let M be the set of tracks with Si = Smax
    Require: Smax % |M| = 0, Snext ≤ Smax*(|M|-1)/|M| (Th. 5)
    Assign all unassigned tracks in M to Oi, s.t. all tracks in M are
      at a k*Smax/|M|, for all k 0≤k<|M|.
    If all tracks have their Oi assigned, end.
    Require: Snext = c*Smax/|M| for some integer c≥1 (Th. 6), and Snext
      % c = 0 (to make Th. 5 work)
    Use c placeholder tracks w/S=Snext to model existing breaks
  }
}
```

Fig. 3. The pseudocode for the Optimal Factor algorithm. This algorithm has been proven optimal [5] provided all restrictions listed in the pseudocode are met.

Using the theorems we have presented, we can construct an algorithm (Fig. 3) which is optimal [5] provided our restrictions are met. There is one additional restriction that is implied. Because the tracks at a given S value must be evenly spread, the offsets assigned to the placeholder tracks added using Theorem 6 in the previous iteration must fall at offsets calculated using Theorem 5 on the next iteration. This is accomplished by requiring that S_{next} also be evenly divisible by the number of pseudotracks of that length added during the track conversion phase.

3.4 Relaxed Factor Algorithm

While the Optimal Factor Algorithm generates placements that are optimal in diversity score, there are significant restrictions on segment length and track quantity.

However, not all architectures may meet these requirements. Therefore we developed a relaxed version, which may not always be optimal, but does demonstrate good results.

The general framework remains basically the same as the optimal version, although the placement and track conversion phases differ in operation. Before we used restrictions on track length and quantity to ensure that our methods are optimal. In the Relaxed Algorithm, when the optimal restrictions are not met, we instead use a number of heuristics where the goal is the even spreading of the breaks in tracks.

The routing architecture can be considered in terms of a topography, where elevation at a given location is equivalent to the number of breaks at that position. Because "mountains" represent areas with many breaks, and therefore fewer potential routing paths, we attempt to avoid their creation. Instead, we focus on placing our tracks to evenly fill the "plains" in our topography, where the breaks from the additional tracks will have a low effect on the overall routability of the resulting architecture. This topography is represented by the breaks[] array, which holds a count of the number of breaks occurring at every position 0…K-1. In this case, K is the number of positions required to observe the full behavior of the break pattern. As mentioned earlier, this value can be determined by finding the LCM of all S values in the problem.

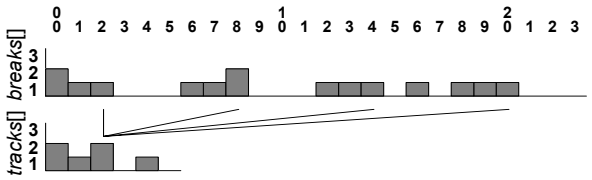

Fig. 4. An example of a breaks[] array for K = 24, and the corresponding tracks[] array for S_{next} = 6. The lines between the two arrays indicate the locations in the breaks[] array used to compute the value of tracks[2].

When each S group is placed, we need to look at the breaks[] array in terms of its effect on that S group. The tracks[] array summarizes the information from the breaks[] array within the potential offsets for a track. Fig. 4 indicates how this is accomplished. Using the tracks[] array, we can choose offsets which place new breaks in slots with the fewest amount of existing breaks. Each time a track is placed, both the breaks[] array and the tracks[] array are updated to reflect the additional breaks. As long as we have at least as many tracks in the current S group as minimum slots in tracks[], this is a simple procedure. However, when there are more minimum locations than tracks, we need to decide which of those minimum offsets should be used.

In that case, we use density-based placement. We first compute the "ideal" density of breaks (ie, if they were placed perfectly smooth) if we were able to achieve a perfect placement of the tracks thus far plus our current S group. This represents our placement goal for the S group. In order to achieve this goal, we consider an increasing region of the array, and attempt to bring its density close to the goal density. This region first begins as a single plain and mountain in the tracks[] array, and the num-

ber of tracks needed to bring the density of that region close to the goal density is added. These tracks are spaced evenly in the plain. The region is now increased to include the next plain and mountain in the array (where we treat tracks[] as a circular array). We then add tracks to the new plain in a similar manner to bring the density of the new region close to the goal density. This continues until the entire tracks[] array is included in the region, and we have placed all our tracks in the current S group.

Once we have placed all tracks with $S = S_{max}$, we need to prepare for the next iteration when we place all tracks with $S=S_{next}$. This means that we need to update the tracks[] array to be in terms of S_{next} instead of S_{max}. The tracks[] array is resized to be S_{next} slots in length, and is recomputed using the technique from Fig. 4.

4 Results

A number of terms are used to describe the problems that we have covered in our testing of our algorithms. The value numT refers to the total number of tracks in a particular track placement problem, numS refers to the number of discrete S values in the problem, maxS is the largest S value in the problem, and maxTS is the maximum number of tracks at any one S value in the problem. We have also reduced our very large search space by only considering problems where the number of tracks at each S value is less than the S value itself, since Theorem 4 strongly implies that cases with S or more tracks of a particular S value will yield similar results by placing tracks from any full sets one per potential offset. Cases with only one track, or all track lengths less than 3, are trivial and thus ignored. The three terms, numT, numS, and maxS, along with the restrictions above, define the track placement problems we consider.

Our first test was to verify that the Optimal Factor Algorithm yields a best possible diversity score in practice as well as theory. The results of this algorithm were compared to those of the Brute Force Algorithm for all cases with $2 \leq numTracks \leq 8$, $1 \leq numS \leq 4$, and $3 \leq maxS \leq 9$, which represents 5236 different routing problems. Note that even with these significant restrictions the runtime of brute-force is over a week of solid computation. In all cases where a solution could be found using the Optimal Factor Algorithm, the resulting diversity score was identical to the Brute Force method. Furthermore, we compared the Relaxed Factor Algorithm to the Optimal Factor Algorithm for this same range and found that Relaxed Factor produces optimal results for all cases that meet the Optimal Factor restrictions within that range.

Next, we compared the performance of the Relaxed and Simple Spread to the results of the Brute Force method for the same search space as above to determine the quality of our algorithms. The results for the heuristics, normalized to Brute Force, are shown categorized by numT, numS, and maxTS in Fig. 5 In these graphs Min is the worst performance of the algorithm at that data point, Max is the best, and Avg is the average performance. In this figure, the Brute Force result is a constant value of 1. Fig. 5 left indicates that both heuristics achieve optimal results in some cases, with the average of the Relaxed algorithm nearly optimal across the entire range. Simple Spread improves with the increasing number of tracks, and both algorithms degrade with an increase in the number of different S values, though only slightly for Relaxed.

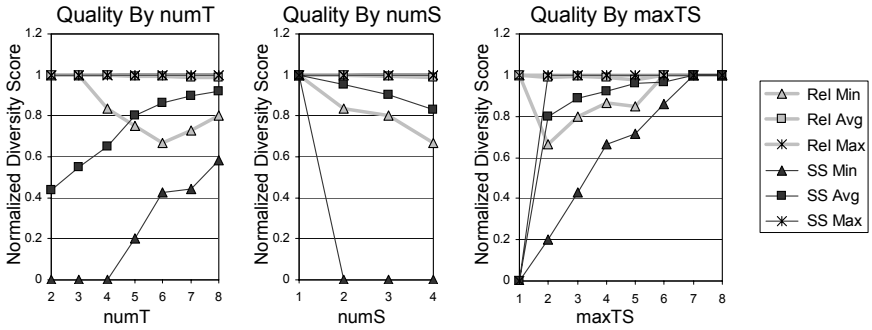

Fig. 5. A comparison of Relaxed and Simple Spread to the Brute Force method, with respect to numT (left), numS (center), and maxTS (right). The diversity scores for each case were first normalized to the Brute Force result, which is represented by a horizontal line at the value 1.

The upswing of both algorithms' minimums towards higher values of numT may be an artifact of our benchmark suite – since we only allow at most 4 unique S values, when there are more than 4 tracks we have at least two tracks with the same S value. Fig. 5 right shows that as the number of tracks per S value increases, the quality of all algorithms improves. The only exception is for the relaxed algorithm when maxTS=1; when there is only one track per S value the Relaxed Algorithm is always optimal. All throughout these tests the Relaxed algorithm is superior to the Simple Spread algorithm. This indicates the critical importance of correlation between S values in track placement. Fig. 5 center also demonstrates how as numS increases, the more difficult it is to solve the problem well, which we expected to be the case. Note that the results for both heuristics are optimal for the case when there is only one S value, as in this case there are no correlations to contend with, and only an even spreading is required.

	OFDM	Camera	Radar	Image Processing	Sort	Matrix Multiply	FIR Filters
Simple Spread	27	23	20	23	23	11	20
Relaxed	24	19	13	15	13	10	11

Fig. 6. The number of tracks in our target architecture required to successfully place and route all netlists in an application using the given track placement algorithm.

Next, we used our place and route tool [1] to test the correspondence between diversity score and routability of architectures created using the Relaxed and Simple Spread algorithms. These architectures are based on a tileable coarse-grained architecture similar in structure to RaPiD. This architecture has two length-2 local tracks, four length-4 local tracks, eight length-4 distance tracks, and eight length-8 distance tracks. Note that local tracks do not allow connections between wire segments to form longer wires. We used seven different multi-netlist applications, and created a test case for each application/track placement algorithm. By keeping the proportion of track types constant but varying the total quantity of tracks, the minimum number

of tracks required to successfully place and route every netlist in the application onto this target architecture was determined. Fig. 6 lists the results of this experiment. Performing track placement using the Relaxed Algorithm allowed netlists to be routed with on average 27% (and up to 45%) fewer tracks than the Simple Spread Algorithm.

5 Conclusions

As we have shown, the track placement problem involved fairly subtle choices, including balancing requirements between tracks of the same length, and between tracks of different, but not relatively prime, lengths. We introduced a quality metric, the diversity score, which captures the impact of track placement. Multiple algorithms were presented to solve the track placement problem. One of these algorithms is provably optimal for some situations, though it is complex and works for only a relatively restricted set of cases. We also developed a relaxed version of the optimal algorithm, which appears to be optimal in all cases meeting the restrictions of the optimal algorithm, and whose average appears near optimal overall.

We envision two situations where this presented research can be applied. First, we believe that there is a growing need for automatic generation of FPGA architectures for systems-on-a-chip. Domain-specific FPGAs can achieve much better area, performance, and power standard FPGAs. Furthermore, because the FPGA subsystem will be within a custom fabricated SoC, the advantage of pre-made silicon of commodity FPGAs is irrelevant. Second, these techniques can be used as a guideline for manual FPGA designers to potentially improve routing architecture quality.

References

1. K. Compton, S. Hauck, “Flexible Routing Architecture Generation for Domain-Specific Reconfigurable Subsystems”, *International Conference on Field-Programmable Logic and Applications*, pp. 59-68, 2002.
2 D. C. Cronquist, P. Franklin, C. Fisher, M. Figueroa, C. Ebeling, "Architecture Design of Reconfigurable Pipelined Datapaths", *Twentieth Anniversary Conference on Advanced Research in VLSI*, 1999.
3. J. R. Hauser, “The Garp Architecture”, *University of California at Berkeley Technical Report*, 1997.
4. V. Betz, J. Rose, ``Automatic Generation of FPGA Routing Architectures from High-Level Descriptions," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 175 – 184, 2000.
5. K. Compton, S. Hauck, “Track Placement: Orchestrating Routing Structures to Maximize Routability”, *University of Washington Technical Report UWEETR-2002-0013*, 2002.