Multi-Million Gate FPGA Physical Design Challenges

Maogang Wang
Cadence Design Systems
mgwang@cadence.com

Abhishek Ranjan
Hier Design Inc
ranjan@hierdesign.com

Salil Raje
Hier Design Inc
salil@hierdesign.com

ABSTRACT

The recent past has seen a tremendous increase in the size of design circuits that can be implemented in a single FPGA. These large design sizes significantly impact cycle time due to design automation software runtimes and an increased number of performance based iterations. New FPGA physical design approaches need to be utilized to alleviate some of these problems. Hierarchical approaches to divide and conquer the design, early estimation tools for design exploration, and physical optimizations are some of the key methodologies that have to be introduced in the FPGA physical design tools. This paper will investigate the loss/benefit in quality of results due to hierarchical approaches and compare and contrast some of the design automation problem formulations and solutions needed for FPGAs versus known standard cell ASIC approaches.

1. INTRODUCTION

Advances in process technology are enabling a profound increase in the number of applications that can be realized using FPGAs. Devices are now being fabricated in advanced, ultra deep sub-micron technology with multi-million gate capacity and clock speed capability approaching 400 MHz. Difficult design problems associated with interconnect delay on large designs are now being seen. As witnessed when high gate-count deep sub-micron ASIC designs first emerged, interconnect can account for as much as 70-90% of overall circuit delay as critical dimensions shrink below 0.18um. These large design sizes also significantly impact cycle time due to software runtimes and an increased number of performance based iterations. EDA tools for FPGA design have failed to keep pace with advances in complex high-density devices. Today, RTL is synthesized using coarse interconnect estimates and then mapped, placed, and routed. If the design fails to meet performance constraints, the designer must make changes to the RTL and/or the constraints and then continually iterate the entire design through the flow.

Hierarchical approaches that partition the design into smaller pieces, implement them separately, and assemble the pieces together in a final assembly phase need to be introduced in the FPGA physical design flow. A hierarchical design flow helps manage the design size complexity for the design team, makes the design flow more predictable, and enables incremental design. Researchers, however, have long claimed that such partitioning of the design leads to sub-optimal results and poor implementations. This paper will quantify the loss in such quality due to design partitioning. The Dragon [24] placement tool framework is used to show our results.

Floorplanning is a key ingredient of the hierarchical approaches. FPGA floorplanning has been tackled by [8]. [8] makes simplifying assumptions about the FPGA architectures. They describe the FPGA architecture as simply rows and columns of CLBs. In this paper, we will describe the new generation FPGA architectures using the Xilinx Virtex-II as a template. The floorplanning problem for these next generation FPGAs is formulated and a desirable cost function is defined in this paper.

Estimation tools at the floorplan level and the placement level are key to reducing the number of iterations done between RTL and the final routed design. Good estimation tools enable a shorter iteration loop between RTL and constraints changes to the floorplan or the placement. This paper will also analyze the variables involved in estimating delays at the placement level.

The paper is organized as follows: Section 2 will give the basics of the Virtex-II architecture. This will form the context for the rest of the paper. In section 3 we will define the floorplanning problem for FPGAs using Virtex-II series as a template. In section 4 we analyze the loss of quality in partitioning the design in smaller pieces to be used in hierarchical design flows. In section 5 we will discuss delay estimation techniques and we will conclude with section 6.

2. FPGA ARCHITECTURE

Till the recent past FPGAs were laid out as rows and columns of CLBs. Recently, however, this rule of square shaped uniform blocks distributed throughout the FPGA fabric has been broken. With greater requirements on the type of logic that needs to be implemented on the same die the FPGA fabric is looking more and more heterogeneous. Figure 2.1 shows the Xilinx xc2v1000 Virtex-II device. The display of the architecture is a logical or an abstract view of the actual device. Each square unit is a CLB. This device has 4 “slices” within each CLB (Configurable Logic Block). Each slice contains 2 4-input LUTs and 2 Flip Flop. RAM and Multiplier blocks are distributed in columns through the device. Each RAM and a Multiplier has a width of half a CLB-column and height of 4 CLB-rows.

VirtexII-PRO is an extension of the Virtex-II architecture and has power PC cores distributed evenly in the FPGA fabric. More extensions are expected to follow which will make this simplifying assumption about a matrix of CLBs less and less accurate.

3. FLOORPLANNING

Floorplanning is a key step in a hierarchical design flow. Floorplanning is performed to get the locations and sizes of the modules to be implemented.

Traditionally the floorplanning problem has been defined to be an area packing problem on a set of modules; some that are hard macros and have fixed height and width, others that are soft modules that have a range of aspect ratios and areas to choose from. Techniques involve having a compact representation of a floorplan [9,10,15,16,17,18,19,20,21] and using simulated

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

annealing moves to modify the floorplan. There has been some work done for fixed outline floorplanning [22] but most of these are extensions of the area packing formulations.

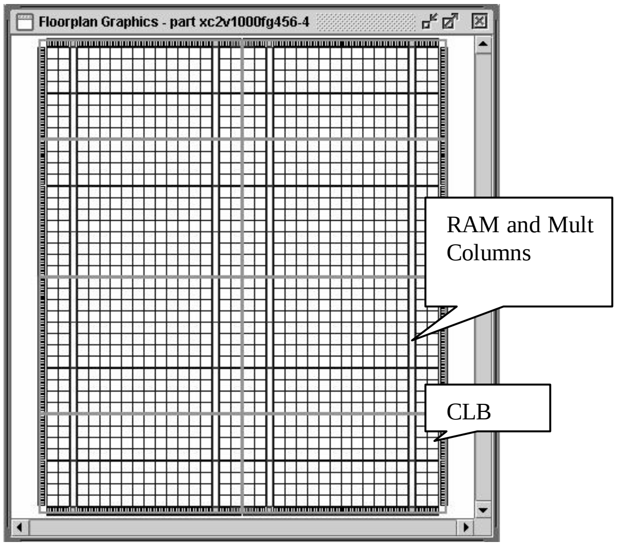

Figure 2.1: xc2v1000 architecture

Floorplanning in the FPGA domain needs a significantly different problem formulation. The FPGA fabric is made up of discrete number of resources n_i of type *i*. The FPGA modules have discrete area requirements. Each module M_j that needs to be placed and shaped would have certain resource requirements such as x_{j1} of type n_1 and x_{j2} of type n_2 and in general x_{ji} of type n_i. The blocks need to be placed and shaped so as to fulfill the resource requirements, fit in the FPGA device with no overlaps and optimize a given cost. Obviously, the traditional floorplan representations and techniques will not be sufficient to meet the needs of the described problem. For one, the traditional floorplan representations assume a continuous space and all locations are available for placement of modules. Secondly, the techniques assume a single area resource requirement that is a continuous function of height and width making the module areas additive.

The cost function to be optimized by the floorplanner needs to account for the interconnect length between modules, the timing properties of the design once its fully implemented through placement and routing, and the routing congestion.

The following subsections will propose a floorplan representation and a "good" cost function for optimization. This paper, however, leaves the actual optimization strategies as future research work.

3.1 Floorplan Representation

The FPGA module floorplan can easily be represented using the location of the lower left corner of the module and the shape of the module.

If, $S = \{(I, J) | 1 \leq I \leq \text{Number of columns in device}, 1 \leq J \leq \text{Number of rows in device}\}$

If M is a set of modules to be floorplanned, then, the set of all possible module placements can be represented as $P = \{(m, l, s) | m \in M, l \in S, s \in S\}$, where m is the module, l is the location of the lower left corner of the module and s is the bounding box height and width of the module. Also, the bounding box for any of the modules defined by l and s needs to be contained within the device boundary. The described representation is not suitable for standard cell ASIC floorplanning because of the continuous nature of the space that can make $|P|$ infinite. As such in the FPGA domain $|P|$ is $O(|M|n^4)$, where n is *max(Number of columns in device , Number of rows in device).*

3.2 Optimization Criteria

The cost function used in standard cell ASIC floorplanning literature is usually a combination of area optimization and connectivity length between modules. The FPGA floorplanning problem is a fixed outline problem and hence area packing is of little consequence. The cost function needs to account for total wirelength (routability) and the timing characteristics of the design. The true measure of a good floorplan can only be ascertained once all modules are physically implemented through place and route. During the actual optimization process, however, the module placement and routing is usually not available. In fact, it is conceivable that even the netlist for these modules are not available at the time of floorplanning. The incompleteness of the design makes defining a "good" cost function during floorplanning a difficult problem.

Generally, it has been observed that given reasonable module partitioning – there are significantly more nets within the module than there are outside the module [6]. Using this observation we claim that it is significantly more important to shape the modules such that there is higher probability of the average internal net lengths to be smaller than the average external net length. The average internal net length can be shortened by making the aspect ratio of the modules as close as possible to 1.0. Secondly, care can be taken during module generation for hierarchical design to latch all the module boundaries such that most timing critical paths after place and route are within the module and do not cross module boundaries.

The above observations lead us to the following cost function:

$$C = \Sigma\alpha|\text{ratio-1}| + \beta(\text{external-wire-length}) + \gamma(\text{overlap})$$

Where ratio is the aspect ratio of the module, external-wire-length is the total wirelength measured as center-to-center Manhattan distance between modules times the number of nets between module pairs, and overlap is the total overlap between pairs of module rectangle boundaries; α, β, and γ are weights used to trade off the different cost criteria. In general, based on the above observations $\gamma >> \alpha >> \beta$. The location and shape that does not satisfy all the resource requirements of a module as explained in the problem formulation is not considered to be legal.

Given the above floorplan problem formulation, representation and a cost function the problem can be solved using a simulated annealing framework, but, we leave the actual optimization technique for further research and scrutiny.

Figure 3.1 shows a sample solution for a design with 7 modules.

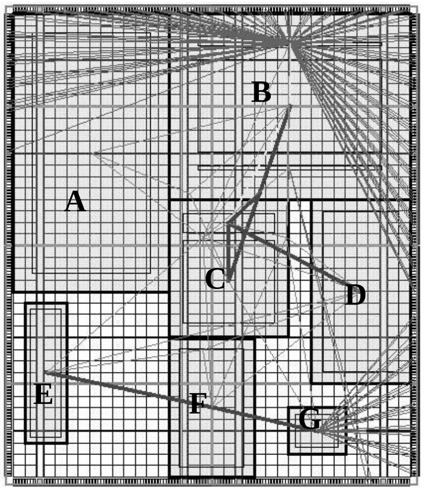

Figure 3.1: Sample Floorplan: Module E is placed far from F to account for the RAM resource requirement on module E.

4. HIERARCHICAL PLACEMENT

Increasing FPGA design size and complexity makes partitioning an important aspect of the physical design flow. Partitioning is the process of dividing the design up into smaller and more manageable modules. The goal of the partitioning step is to minimize the interdependence between the partitioned modules. Once partitioned, each of the modules are then optimized independently of each other and assembled together in a final assembly phase [2, 23]. Naturally, the smaller the size of the partitions and more dependence they have on each other the greater the loss in quality of results as compared to optimizing the design in its entirety. This section quantifies this very loss in quality due to partitioning. The framework used to study partitioning and its effect on the final quality of results is a hierarchical placement tool that mimics just such partitioning techniques [24, 26, 27].

4.1 Overview of Hierarchical Placement

The hierarchical placement approach is essentially a divide-and-conquer scheme. At each hierarchical level, the original placement problem will be divided into several sub-problems. Heuristic algorithms are then used to solve each of the sub-problems. In this approach, the sub-problems never interact with each other again. The final placement solution is just the union of solutions collected from all the solved sub-problems. For example, assume at the highest hierarchical level, the placer divides the original netlist into four parts: A, B, C and D and assigns them to 4 disjoint regions R_A, R_B, R_C and R_D respectively. If a cell is assigned to A and R_A at this stage, it will never appear in the physical regions: R_B, R_C or R_D.

Divide-and-conquer is one of the most popular placement approach, however, Dragon [24], a high-performance ASIC hierarchical placer, breaks away from this mould in that it does allow a cell which was originally assigned to R_A to be moved to R_B, R_C or R_D. The following sections use Dragon to study the effects of partitioning on the quality of placement.

4.2 Partition and Placement in Hierarchy

There are two critical parameters that affect the quality of results in a hierarchical placement approach: 1). Partitioning the netlist into smaller sub-netlists. 2). Assigning these partitions to physical regions. We try to study the relations between these two aspects and the impact they have on the final placement quality. Our study can provide a deeper insight for developing next generation placement tools as well as provide insight into the loss of quality incurred through hierarchical design flows for multi-million gate FPGA designs.

4.3 Partition Quality vs. Placement Quality

As indicated in section 4.2, how to perform netlist partitioning and how to place partitioned cell clusters are two most important aspects in the hierarchical placement context. In this sub-section, we designed series of experiments to try to reveal the relationship between these two aspects and the final placement quality.

We use Dragon as the test-bed to perform our experiments as Dragon is very flexible in adding other components to mimic the behaviors of different placement approaches.

Hierarchical placement algorithms work in a multi-level context. Interactions between hierarchical levels also affect the final placement quality. In order to focus solely on the two aspects pointed out in section 4.2, we designed controlled experiment which can filter out the interactions between levels.

We let our test system behave as if there is only one level of hierarchy in the placer. The test placer will divide the original netlist into a number of sub-netlists. After that we let our placer solve placement problems for each sub-netlist. The final placement is obtained by assembling placement results for these sub-netlists together onto the original layout area. We test three cases which divide the original netlist into 4, 16 or 64 sub-netlists, respectively.

For each case, we have 5 different partitioning schemes, Par1, Par2, Par3, Par4 and Par5, each of them with a different partitioning quality. These five schemes are ordered from the highest result quality to the lowest result quality. Par1 is hMetis [13] which is one of best existing min-cut partitioners. Other four schemes are created by adding some artificial perturbations to the partitioning results obtained from Par1. Specifically, Par2 randomly exchanges 0.5% of cells, Par3 randomly exchanges 1% of cells, Par4 randomly exchanges 5% of cells and Par5 randomly exchanges 10% of cells between partitions.

In addition to these 5 partitioning schemes, we tested 3 cluster placement schemes, CP1, CP2 and CP3, where CP1 being the highest quality scheme and CP3 being the lowest quality scheme. CP1 is the regular simulated annealing cluster placement method used by Dragon. CP2 is a greedy cluster placement method and CP3 is a simulated annealing method which intentionally tries to increase the cluster wirelength.

We can mimic the behavior of a hierarchical placement approach by pairing one partitioning scheme with one cluster placement scheme. Thus we have a total of 15 configurations with each of them being represented by pair {Par[1-5], CP[1-3]}. We picked three circuits from the IBM placement benchmark suite to run our experiments. Table 4.1 shows the properties of these three circuits.

Table 4.1. Testing circuit statistics (picked from IBM placement benchmark suite).

circuits	cells	nets	rows	white space	core(row) util.	routing layers
ibm01	12,028	11,753	132	14.88%	85.12%	4
ibm08	50,672	48,230	243	9.97%	90.03%	5
ibm12	68735	68,376	347	14.78%	85.22%	5

Table 4.2 shows the 4-partition placement results for all 15 configurations. The results are normalized with the results generated by Dragon. Table 4.3 and 4.4 show the placement results for the 16-partition and 64-partition case, respectively.

Table 4.2. One-level hierarchical 4-partition placement results comparison (numbers in the table are final total wirelength normalized to the results obtained from Dragon).

4-partition		Par1	Par2	Par3	Par4	Par5
IBM01	CP1	1.008	1.026	1.109	1.428	1.871
	CP2	1.006	1.025	1.108	1.445	1.798
	CP3	1.053	1.087	1.235	1.749	2.351
IBM08	CP1	1.026	1.072	1.142	1.358	1.590
	CP2	1.034	1.051	1.068	1.329	1.583
	CP3	1.129	1.127	1.338	2.117	3.015
IBM12	CP1	1.020	1.056	1.140	1.554	2.019
	CP2	1.025	1.057	1.153	1.517	2.092
	CP3	1.076	1.123	1.261	2.063	2.908

Table 4.3. One-level hierarchical 16-partition placement results comparison.

16-partition		Par1	Par2	Par3	Par4	Par5
IBM01	CP1	1.038	1.030	1.100	1.196	1.307
	CP2	1.056	1.060	1.084	1.164	1.384
	CP3	1.907	1.817	2.007	2.727	3.517
IBM08	CP1	1.035	1.107	1.080	1.242	1.307
	CP2	1.055	1.088	1.111	1.213	1.338
	CP3	1.756	1.897	2.028	3.076	3.985
IBM12	CP1	1.054	1.171	1.153	1.278	1.404
	CP2	1.119	1.108	1.095	1.246	1.369
	CP3	1.912	1.811	2.078	3.111	4.138

4.4 Observations

The experimental data supports and validates a number of intuitions. There are several important observations we can make from the experimental data.

1). We compare results obtained using the same configuration but different circuits. They are consistent to a certain extent. However, there are discrepancies which are not negligible. For example, the configuration {Par2, CP1} in Table 4.3 are quite different for three circuits (1.030 vs. 1.107 vs. 1.171). Thus different connectivity configurations have impacts on the performance of a certain hierarchical placement algorithm. A good placement tool should be adaptable based on the circuits' topological information [25].

Table 4.4. One-level hierarchical 64-partition placement results comparison.

64-partition		Par1	Par2	Par3	Par4	Par5
IBM01	CP1	1.088	1.121	1.100	1.173	1.216
	CP2	1.156	1.145	1.169	1.180	1.258
	CP3	2.978	3.050	3.138	3.831	4.619
IBM08	CP1	1.165	1.129	1.147	1.180	1.269
	CP2	1.159	1.235	1.228	1.265	1.342
	CP3	2.953	2.953	3.232	4.230	5.360
IBM12	CP1	1.162	1.151	1.193	1.293	1.332
	CP2	1.195	1.172	1.218	1.291	1.282
	CP3	3.052	3.072	3.327	4.357	5.351

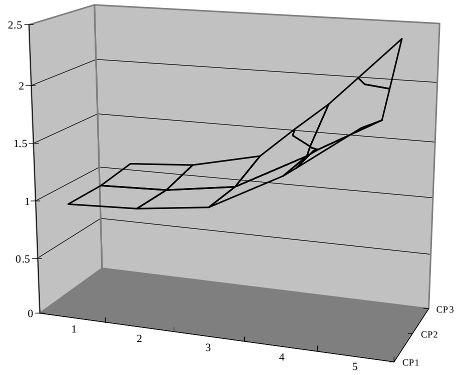

Figure 4.1. Normalized 4-partition placement wirelength obtained by 15 different hierarchical configurations. Each configuration is represented by pairing one of the 5 partitioning schemes (x-axis) and one of the 3 cluster placement schemes (y-axis).

2). The final placement wirelength degrades when a worse partitioning scheme is used.

The placement results degrade from Par1 to Par5 for all circuits tested. With 1% of cells being randomly swapped between partitions (Par3), the final placement results can be degraded by roughly 10%. Fig. 4.1 plots the normalized 4-partition placement wirelength for all 15 configurations in a 3D view. From this observation, we can argue that improving partitioning quality is crucial in all min-cut based hierarchical placement tools. For other tools which perform partitioning based on physical cell locations obtained from analytical methods, probably more considerations on minimizing interconnections between clusters will also be helpful.

3). The final placement wirelength degrades when a bad cluster placement scheme is used.

If we compare results using the same partitioning scheme but different cluster placement schemes, the one obtained by the bad cluster placement scheme (CP3) is obviously worse than the other two schemes. In majority cases, the final placement result of CP1 is better than the result of CP2. This observation shows the cluster placement quality correlates with the final placement wirelength in general. However, there are several counter examples where CP2 outperforms CP1. They will discussed later in this sub-section.

4). Recursive divide-and-conquer approaches indeed have a quality loss over "flat" placement approaches.

Partitioning scheme Par1 and cluster placement scheme CP1 are the same schemes used inside Dragon. The only difference between Dragon and configuration {Par1, CP1} is that {Par1, CP1} lets cells remains in their partition while Dragon does not have this restriction. Thus the comparison between the results of Dragon and the results of {Par1, CP1} represents the quality loss by imposing the divide-and-conquer restriction in the hierarchical context. From the data shown in Table 4.2, we find that the final wirelength is roughly increased by 2% for the 4-partition case. The quality loss (wirelength increase) is higher for the 16 and 64-partition case (roughly 4% and 12% respectively).

Please note our test system only has one level of hierarchy. For a multi-million gate design, the placement tools have around 10 hierarchical levels if the quadrisection method is used. Thus it is reasonable to believe that a hierarchical placement tool will have a quality loss of 10-20% over a "flat" placement tool under the assumption that the runtime of the “flat” placement tool is not a concern.

5). Partitioning quality affects final placement results less and less when the number of partitions increases.

We compare results from {Par5, CP1} and results from {Par1, CP1}. In Table 4.2 (4-partition case), {Par5, CP1} produces results roughly 80% worse than {Par1, CP1} does, while in Table 4.3 (16-partition case) and Table 4.4 (64-partition case), this difference is only 33% and 27%, respectively.

This is because partitioning correlates with placement better when the number of partitions is small. Thus in the 4-partition case, a bad partitioning scheme (Par5) would severely affect the final placement result. In the 16-partition and 64-partition case, while using a better partitioning scheme can still help improve the final placement (Observation 2), it plays a less important role than in the 4-partition case.

6).A better wirelength at a hierarchical level does not necessarily imply a better final placement wirelength.

This observation is made by the fact that there are a number of places where CP2 outperforms CP1 while CP1 is a better cluster placement scheme than CP2. Due to the limitation of our test system and the nature of our partitioning schemes (randomized methods), even with the same partitioning scheme picked, different CP schemes most likely have non-identical initial clusters/partitions. Thus the numbers shown in the tables have some degrees of uncertainty. Nevertheless, it shows a better wirelength in a hierarchical level does not guarantee a better final wirelength.

While this is more or less a disappointing observation for a placement problem researcher, there is one thing needed to be noticed. When a good partitioning scheme is used, hierarchical wirelength correlates better with the final wirelength. Most of the “bad” correlation happens when a “bad” partitioning scheme is used.

7). Different partition sizes have trade-offs between each other.

Three cases are tested in this section, 4-partition, 16-partition and 64-partition. Each of them has its own advantages and disadvantages. When the number of partitions is small at a given hierarchical level, there is a better correlation between partitioning and placement and the cluster placement problem is almost trivial. The disadvantage is that more hierarchical levels are required and there is a quality loss at each level. When the number of partitions is large, there is a better correlation between the cluster wirelength and the final wirelength and a fewer number of total hierarchy is needed. However, getting a good clustering and a good cluster placement becomes a non-trivial problem.

5. DELAY ESTIMATION

Delay estimation for FPGAs lacks a coherent methodology. Several recent works have tried to estimate wire delays at various levels of design stages [4], [5], [11], [12]. Almost all of these previous works have exploited features (routing resources, gridded architecture etc.) of targeted FPGA architecture to estimate delays. Though relevant to the targeted FPGA, most of these previous methodologies can not be applied to recent multi-million gate FPGAs because of the complexity of the estimation process [11][12].

Delay estimation has three basic types [1]: *a priori*: estimating delays before placement and routing, *a posteriori*: for a given placement estimate the routing delays and *on-line*: estimating delays as the placement or routing is going on. In this work we will concentrate on *a posteriori* delay estimation. The models developed for *a posteriori* delay estimation can easily be extended to other types of estimation as well.

Terminologies and Experimental Setup:

From now on whenever we say *distance* we mean Manhattan distance between the CLBs in which driver and driven pins are respectively located. Similarly, *delay* would mean delay between driver and driven pins.

We will show results on six (multi-million gate) industry designs. Details of the benchmarks are in Table 5.1. All the data for these experiments was generated using Xilinx’s Place and Route tool, PAR [28]. PAR was run in high-effort mode for timing and routing optimization.

Variables affecting wire delays:

Some of the variables which traditionally have been explored for wire delay estimation are: Fanout of nets (connecting driver-driven pin pairs), distance between driver-driven pin pairs and routing congestion [12]. Of these, routing congestion is the hardest to measure and depends heavily on routing algorithm being used. Congestion estimation is beyond the scope of this work. We will limit ourselves to studying the impact of net fanout and distance between driver-driven pin pairs on wire delays.

After studying the effectiveness of these, we will propose a more realistic variable, *number and types of routes* (which most of the previous works have overlooked), and show that it correlates really well with the delay of pin pair under consideration.

Table 5.1. Details of the benchmarks

Design	Number of gates	Targeted FPGA (Virtex-II)	Device Size (CLBs)
ind_ccd	168K	2v500	32x24
ind_st	1.6M	2v1000	40x32
ind_ink	1.3M	2v1000	40x32
ind_e5k	9.9M	2v6000	96x88
ind_apa	7.5M	2v6000	96x88
ind_com	8.7M	2v8000	112x104

Fanout of the net connecting pin pair

Fanout of the net has been shown to correlate really well with the delay in Standard-Cell design methodologies and has been used extensively to derive wire-load models for net delays [3]. However, our experiments show that in the FPGA domain a very weak correlation exists between the two. In Figure 5.1, we show post-routing delay for pin pair (with a distance of 2) versus net fanout plot for ind_com design. For different values of the fanout the range of delays is almost same, giving rise to the notion that fanout has very little impact on delay. The reason why fanout of the net does not impact delay lies more in the routing architecture of the Xilinx FPGA devices. These devices use buffered inter-connects to route the nets [28]. Routing switches break the net at regular intervals and hence the traditional fanout based wire-load models, using *Elmore delay* [7], cease to work. We will describe the routing architecture of these devices in the next a few sub-sections.

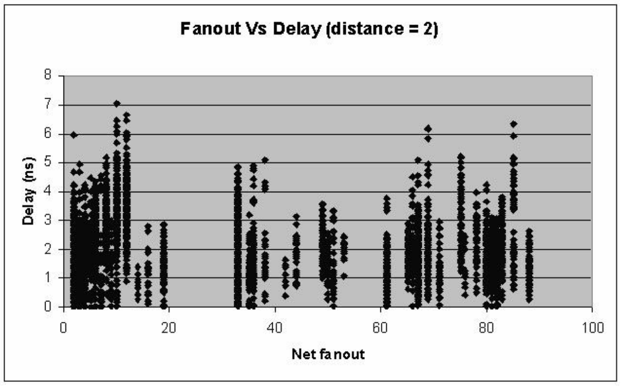

Figure 5.1. Net fanout Vs delay for a fixed distance.

Distance between pin pairs

Authors in [12] have shown that delay between a pin pair has no direct linear relation-ship with the distance between them. To corroborate their observation, for every driver-driven pin pair (for nets with fanout of 2) in ind_com design we plot distance versus post-routing delay. It is evident from the plot, Figure 5.2, that even though the delay seems to be increasing with the distance, for a given distance the range of delays is too large (similar results were seen for other testcases). Trying to fit a linear line to extrapolate the delay for a given distance will have huge error margin.

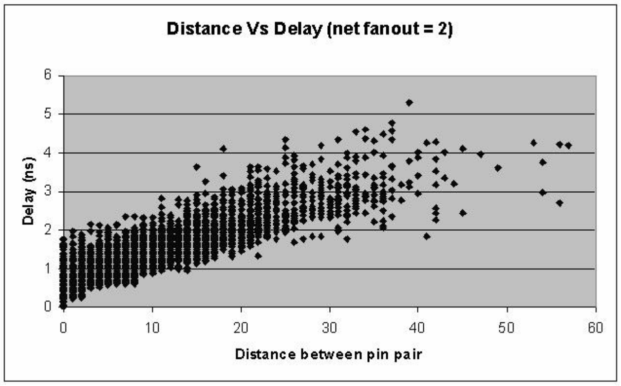

Figure 5.2. Distance Vs delay for a fixed fanout.

However, if we plot delays for a given distance, interesting patterns emerge. Figure 5.3 shows the plots of delays of all the driver-driven pin pairs in ind_com design with a distance of 2. We see that most of the delays are centered around very few vertical lines thereby indicating that delays are combination of discrete values.

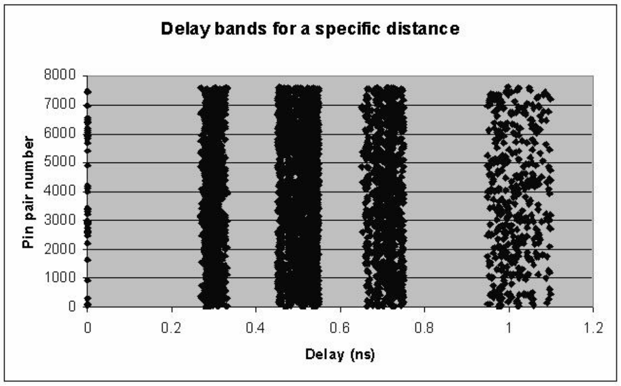

Figure 5.3. Discrete delay bands for a distance of 2.

To motivate our argument for a discrete delay model, it would help to give an overview of routing architecture in Xilinx Virtex-II devices, details of which can be found in [28].

Figure 5.4 (squares denote CLBs and rectangles are slices) shows types of routes that occur in Virtex-II family of Xilinx devices. *Long lines* span full height and width of the chip, *Hex lines* route signals to every third or sixth CLB in all four directions, *Double lines* route signals to every first or second CLB in all four directions, *Direct lines* connect signals to neighboring blocks and the *Fast lines* are internal CLB local connections.

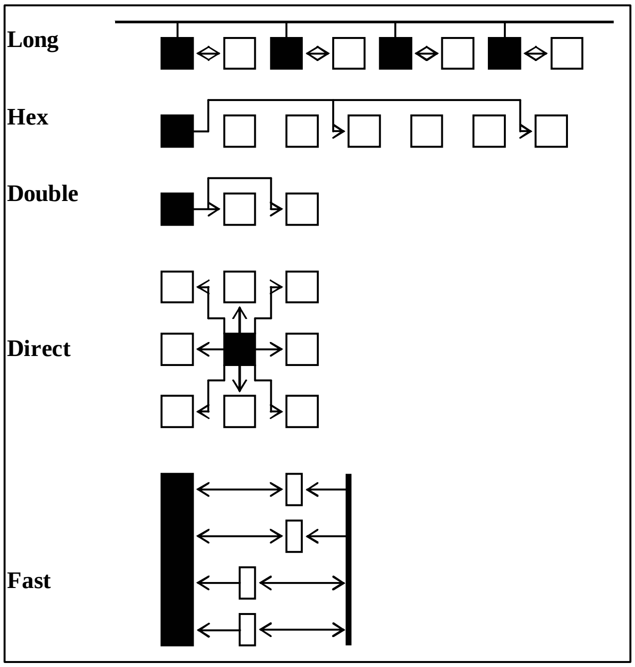

Figure 5.4. Routing lines in Virtex-II devices.

It is this discrete routing structure which gives rise to vertical lines in Figure 5.3. For a distance of 2 CLBs between a driver-driven pair, the number of possible routes is limited. Such a distance can be routed using either of the following: one double line, two double lines, or a direct line and a double line etc. The delays for these different types of lines are almost constant (minor variations might occur due to switch-box delays) and have no relation to each other. For example, delay of a hex line is not three times the delay of a double line or six times the delay of a direct line but is only slightly larger than the delay of a double or direct line.

Hence given a placement, the delay estimation problem gets reduced to estimate the number and types of routes that will be used to route a particular driver-driven pin pair.

Number and types of routes used to connect pin pair

Given the placement for two pins, estimating the number and types of routes is a non-trivial problem. One has to predict almost exactly how the detailed router routes this connection. A good timing driven router will try to route driver-driven pin pairs on critical paths using as less routing lines as possible. We rely on this assumption of the router (use longest line first) to come up with our delay estimation algorithm. Let us denote delays of different lines by D_{long} , D_{hex} , D_{double} , D_{direct} and D_{fast} respectively. Our algorithm for estimating delay between a driver-driven pair is given below.

For each of the six placed and routed benchmark designs we estimate delays for all the driver-driven pin pairs in the design using algorithm described above. To show the quality of our estimation, for ind_com design, we have plotted the difference between the estimated delay and the post-routing delay as a histogram in Figure 5.5. X-axis of the plot is the difference in delays and y-axis is the number of driver-driven pairs with that difference. Ideally we would like the histogram-curve to be a straight line at zero.

```
estimate_driver_to_driven_delay( clbDriver, clbDriven ) {
   if ( distance_between( clbDriver, clbDriven ) == 0 ) {
      fast  = 1;
   } else {
     hori = horizontal_distance( clbDriver, clbDriven );
     vert = vertical_distance( clbDriver, clbDriven );
     get_num_lines( hori, long, hex, double, direct );
     get_num_lines( vert, long, hex, double, direct );
   }
delay =  $D_{long}$*long +  $D_{hex}$*hex +  $D_{double}$*double +
            $D_{direct}$*direct +  $D_{fast}$*fast;
}
get_num_lines( dist, long, hex, double, direct ) {
   long += dist/length_of_long;
   dist = dist%length_of_long;
   hex += dist/length_of_hex;
   dist = dist%length_of_hex;
   double += dist/length_of_double;
   dist = dist%length_of_double;
   direct = dist;
}
```

Even with such a simple delay estimation algorithm, our histogram-curve is very close to perfect. Most of the driver-driven delay estimates are within 1.0ns of corresponding post-routing delays. One more feature of the curve is that they are evenly spread around zero. So for a timing path with long logic depth, the error in delay estimation might eventually cancel out.

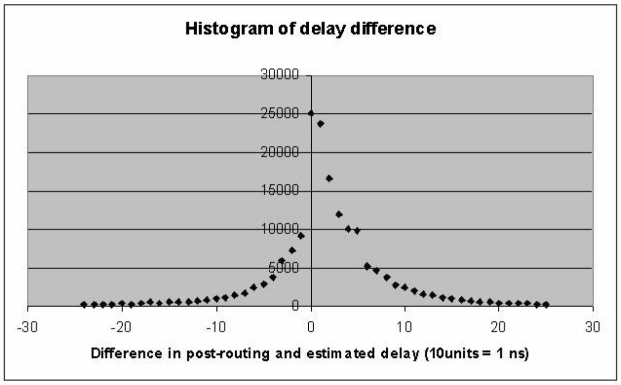

Figure 5.5. Histogram of difference in estimated and post-routing delays.

We also report the estimated longest critical path delay versus the actual post-routing path delay. Results are show in the Table 5.2 below. Our estimated path delay differs from the actual post-routing path delay by 6% on average. It shows that our new estimation method is very effective at the placement level.

Table 5.2. Estimated Vs Post-routing delays.

Design	Estimated path delay (ns)	Post-routing path delay (ns)	% error
ind_ccd	8.50	9.17	7.30%
ind_st	17.17	17.14	0.20%
ind_ink	7.45	7.00	6.43%
ind_e5k	19.35	23.12	16.3%
ind_apa	13.63	14.43	5.54%
ind_com	14.76	14.66	0.68%

6. CONCLUSION

FPGA design sizes have seen tremendous growth over the past few years. It is clear that hierarchical design methodologies need to be introduced to handle the design size and complexity in the FPGA domain. Most research work till date has focused mostly on standard cell ASIC design. We have shown that the problem formulations and solutions designed in the ASIC domain cannot necessarily be transported and made to work in the FPGA domain.

We have studied 3 key ingredients of a hierarchical flow – Floorplanning, hierarchical placement, and delay estimation. We have formulated the FPGA floorplanning problem and described a floorplan representation and proposed a cost function. The actual floorplan optimization techniques need to be researched and is left as future work.

We have studied the impact of partitioning and cluster placement on the final placement. We conclude that the partitioning quality is critically important for producing a high quality final placement. We also show that the divide-and-conquer scheme will be likely to have a quality loss of 10-20% comparing with an ideal "flag" placement approach.

We have explored the traditional variables that impact delay estimation. Variables such as fanout and Manhattan distance that are used extensively in the standard cell ASIC domain cease to work in the FPGA domain. We have proposed new variables – number and types of routes – to estimate post routing delay at the placement level. Results of our estimation algorithm correlate very well with the post routing delay numbers and the average error in prediction is within 6%.

7. REFERENCES

[1] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-based Placement," ISPD, pp. 4-11, 1998.

[2] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" in Design Automation Conference, pp. 477-482, 2000

[3] C. Chen and C. Tsui, "Timing Optimization of Logic Network using Gate Duplication", ASP-DAC, pp. 233-236, 1999.

[4] C. S. Chen, Y.-W. Tsay, T. HwangA. C. H. Wu and Y.-L. Lin, "Combining Technology Mapping and Placement for Delay-Optimization in FPGA Designs," ICCAD, pp. 240-247, 1993.

[5] J. Cong and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," ICCAD, pp. 48-53, 1992.

[6] W. E. Donath, "Placement and average interconnection lengths of computer logic", IEEE Transactions on Circuits and Systems, CAS-26(4):272-277, April 1979.

[7] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wide Band Amplifiers," J. Applied Physics, 19(1), 1948.

[8] J. M. Emmert, and D. Bhatia, "A Methodology for Fast FPGA Floorplanning", Proc. FPGA, 1999.

[9] P. N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-Tree Representation of Non-Slicing Floorplan and Its Applications", Proc. DAC, pp. 268-273, 1999.

[10] M. Z. Kang and W. Dai., "Arbitrary Rectilinear Block Packing Based on Sequence Pair", Proc. ICCAD, pp. 259-266, 1998

[11] T. Karnik, "Hierarchical Timing-Driven Partitioning and Placement for Symmetrical FPGAs," PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[12] T. Karnik and S. M. Kang, "An Empirical Model For Accurate Estimation of Routing Delay in FPGAs," ICCAD, pp. 328-331, 1995.

[13] G. Karypis and V. Kumar, "Multilevel k-way Hyper-graph Partitioning", in Design Automation Conference, pp. 343-348, 1999

[14] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", IEEE Transactions on Computer Aided Design, 10(3): 365-370, 1991

[15] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence Pair", IEEE Trans. On CAD, vol 15(12), pp. 1518-1524, 1996

[16] R. Nair, C. L. Berman, P. Hauge, E. Yoffa, "Generation of Performance Constraints for Layout", IEEE Trans. On CAD, vol. 8(8), pp. 860-874, 1989.

[17] R. H. J. M. Otten, "Efficient Floorplan Optimization", ICCD, pp. 499-503, IEEE/ACM, 1983

[18] R. H. J. M. Otten, "Automatic Floorplan Design", Proc. DAC, pp.261-267, 1992

[19] L. Stockmeyer, "Optimal Orientation of Cells in Slicing Floorplan Designs", Information and Control,57(2), pp. 91-101, 1983

[20] X. Tang, R. Tian and D. F. Wong, "Fast Evaluation of Sequence Pair in Block Placement by Longest Common Subsequence Computation", DATE 2000, pp. 106-111.

[21] X. Tang and D. F. Wong, "FAST-SP: A Fast Algorithm for Block Placement Based on Sequence Pair", ASPDAC 2001.

[22] A. Ranjan, K. Bazargan, M. Sarrafzadeh, "Fast Hierarchical Floorplanning with Congestion and Timing Control", IEEE International Conference on Computer Design (ICCD), pp. 357-362, September 2000.

[23] J. Vygen, "Algorithms For Large-scale Flat Placement", Proc. Design Automation Conference, pp. 746-51, 1997.

[24] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Standard-Cell Placement Tool For Large Industry Circuits", ICCAD, pp. 160-163, 2000

[25] H. Xu, M. Wang, B. Choi and M. Sarrafzadeh, "A Trade-Off Oriented Placement Tool", ICCAD, 2002

[26] X. Yang, B. Choi and M. Sarrafzadeh, "A Standard-Cell Placement Tool for Designs with High Row Utilization", ICCD, pp. 45-47, 2002

[27] X. Yang, B. Choi and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement", ISPD, 2002

[28] Xilinx Inc., http://www.xilinx.com/