UC Irvine

ICS Technical Reports

Title

Area and timing estimation for lookup table based FPGAs

Permalink

<https://escholarship.org/uc/item/8tm5q7xc>

Authors

Xu, Min

Kurdahi, Fadi

Publication Date

1995-08-20

Peer reviewed

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Area and Timing Estimation for Lookup Table Based FPGAs

Min Xu†
Fadi Kurdahi‡

Technical Report #95-30
Aug. 20, 1995

†Department of Information and Computer Science
‡Department of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92717-3425
(714) 824-8104

mxu@ics.uci.edu
kurdahi@ece.uci.edu

Abstract

The importance of efficient area and timing estimation techniques is well-established in High-Level Synthesis (HLS), since it allows more efficient exploration of the design space while providing HLS tools with the capability of predicting the effects of technology-specific tools on the design space. Much of previous work has focused on estimation techniques that use very simple cost models based solely on the gate and/or literal count. Those models are not accurate enough to allow effective design space exploration since the effects of interconnect can indeed dominate the final design cost. The situation becomes even worse when the design is targeted to Field Programmable Gate Array (FPGA) technologies since in most FPGA designs, the wire delay may contribute up to 60% of the overall design delay. In this paper, we present an approach of estimating area and timing for lookup table based FPGAs which uses a realistic model since it takes into account not only gate area and delay but also the wiring effects. We select Xilinx XC4000 series as our main concentration because of their popularity. Our approach is easy to adapt to other similar lookup table based FPGAs. We tested our estimator with several benchmarks and the results show that we can get accurate area and timing estimates efficiently.

SLBAR

Z

699

C3

no. 95-30

Notwithstanding this Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Contents

1	Introduction	4
2	Overview of Xilinx XC4000	5
2.1	XC4000 Configurable Logic Blocks and Lookup Tables	5
2.2	XC4000 Programmable Interconnect Point and Routing Resources	6
3	Previous Work	8
4	Estimation for LUT Based FPGAs	9
4.1	Problem Definition	9
4.2	Deficiencies of Standard Area Estimation Models	9
4.3	Wiring Delay in FPGAs	11
4.4	Area Estimation Approach	11
4.4.1	Predicting LUT Mapping	11
4.4.2	Predicting CLB Construction	13
4.4.3	Predicting Placement and Shape	13
4.5	Timing Estimation Approach	18
4.5.1	Predict CLB Delay	20
4.5.2	Predict Wiring Delay	20
4.5.3	Predict Input-to-output Worst Case Delay	24
4.5.4	Predict Timing Optimization	24
5	Experimental Results	25
5.1	Experimental Procedure	25
5.2	Results	25
6	Conclusion	26
7	References	26

List of Figures

1	The importance of estimation in a typical HLS design flow targeted for FPGAs (a) design flow without estimation (b) design flow with estimation. .	4
2	Xilinx XC4000 architecture (a) XC4000 (b) abbreviated CLB architecture (c) a LUT implementing $x = mn + \bar{n}p$	6
3	One XC4000 CLB can not implement all the functions with less than 10 distinct inputs (a) 9 literals, 9 distinct inputs need 1 CLB (b) 9 literals, 6 distinct inputs need 2 CLBs.	7
4	Wiring architecture (a)single length lines (b)double length lines (c) the XC4000 switch metric. connections and PIP.	7
5	Gate/input count is not a good measure of LUT count (a) 8 gates, 3 inputs need 1 LUT (b) 3 gates, 4 inputs need 2 LUTs (c) 6 gates, 4 inputs need 3 LUTs.	9
6	Number of CLB is not constrained by number of LUT (a) 3 LUTs need 1 CLB (b) 3 LUTs need 2 CLB.	10
7	Net length does not necessarily correlate well with performance (a) same Manhattan distance connections with different delays (b) different Manhattan distance connections with same delays.	10
8	Area estimation approach.	11
9	A walk-through example (a) predicting LUT mapping (b) predicting CLB construction (c) predicting placement and shape.	12
10	Different CLB configurations.	13
11	Lookup table mapping algorithm.	14
12	Lookup table generate algorithm.	15
13	CLB generate algorithm.	16
14	An Example with 9 CLBs (a) slicing tree and the placement(asp = 1) (b) slicing tree and the the placement (asp = 2.5).	17
15	Wire budget (a) with horizontal slice line (b) with vertical slice line.	18
16	Approximate topology for apex7.	19
17	Timing estimation approach.	19
18	The parameters' value for different kind of CLB outputs.	20
19	Point-to-point delay model and associated parameters.	21
20	Routing rule algorithm.	22

21	Delay Algorithm.	23
22	The parameters for the delay model and timing optimization.	23
23	Experiment results (a) unoptimized results (b) optimized results.	26

1 Introduction

The ability to shorten development cycles has made Filed Programmable Gate Array(FPGA) an attractive alternative to standard cells and Mask Programmed Gate Arrays (MPGAs) for the realization of Application-Specific Integrated Circuits (ASICs). High Level Synthesis (HLS), on the other hand, is becoming the methodology of choice for shortening the design time by allowing the user to start from a behavioral specification. Thus, the marriage of these two concepts provides an ideal testbed for fast prototyping from an idea to a final product.

Figure 1: The importance of estimation in a typical HLS design flow targeted for FPGAs (a) design flow without estimation (b) design flow with estimation.

HLS generates an architecture from a behavioral specification subject to constraints on area and delay. Following that, the design process of FPGAs can be decomposed into four major steps as shown in Figure 1(a). First, **partitioning** (or **technology mapping**), which includes Lookup Table (LUT) mapping and Configurable Logic Block (CLB) construction, partitions the incoming logic into a netlist of CLBs. Following that, **placement** determines a good assignment for the CLBs in the FPGA array. Once the placement is known, **routing** decides the type of routing resources and route for each net. Finally, **timing optimization** improves the performance of the placement by re-routing the nets on the critical path.

These four steps make the design very unpredictable and the resultant design may violate the constraints. The reason is that in most FPGA designs, the wire delay, which is not considered in HLS, may contribute up to 60% of the overall design delay. The problem becomes especially acute when the design process starts at the behavioral level using HLS.

In this case, a large number of candidate RTL designs are generated and must be evaluated to select the best design. Abstract cost measures which do not consider layout effects are likely to result in suboptimal designs. Thus, the design process may have to go through several iterations to reach an acceptable solution. Since placement and routing are usually quite time consuming, this may offset any turnaround time advantages of FPGAs and HLS. Indeed, such common situations have been reported in [1]. To avoid unnecessary iterations and shorten the design cycle, it is very helpful to have an estimator giving area and timing estimates quickly before actually going through all the time consuming physical design phases as shown in Figure 1(b). It is very important that the estimator has a more realistic and accurate model which takes into account not only gate area and timing, but also wiring effects.

One important class of FPGAs, implemented by Xilinx, uses LUTs to implement combinational logic and is called LUT based FPGAs. Xilinx has three logic cell array families of LUT based FPGAs including XC2000, XC3000 and XC4000. They share a common structure: an array of CLBs surrounded by configurable interconnect and they differ in details of the logic and interconnect structures. In this paper, we will concentrate on area and timing estimation for LUT based FPGAs. To be specific, we target the Xilinx XC4000 series because of their popularity. As stated before, our intended application domain is HLS since this is where fast and accurate estimation is most needed to support a high quality rapid prototyping environment. Our estimation can be applied either at the chip level where the entire design is translated into a logic netlist before physical design, or at the component level where estimation of different RT level components are needed to support a true hierarchical methodology [2].

2 Overview of Xilinx XC4000

Xilinx XC4000 consists of an array of CLBs embedded in a configurable interconnect structure and surrounded by configurable I/O blocks as shown in Figure 2(a). The Xilinx XC4000 family consists of ten members. The family members differ in the number of CLBs, (ranging from 8x8 to 24x24), and I/O blocks, (ranging from 64 to 192). The typical gate capacity varies from 2000 to 13000.

2.1 XC4000 Configurable Logic Blocks and Lookup Tables

Figure 2: Xilinx XC4000 architecture (a) XC4000 (b) abbreviated CLB architecture (c) a LUT implementing $x = mn + \bar{n}p$.

Xilinx XC4000 CLBs mainly consist of two 4-input LUTs, which are called F-LUT and G-LUT respectively, and one 3-input LUT, which is called H-LUT as shown in Figure 2(b). A K -input LUT is a memory that can implement any Boolean function of K variables. The K inputs are used to address a $2^K \times 1$ -bit memory that stores the truth table of the Boolean function. For example, Figure 2(c) illustrates the structure of a 3-input LUT for implementing function $x = mn + \bar{n}p$. The truth table of the function is stored in an 8x1-bit memory, and an 8 to 1 multiplexor, controlled by the variables m, n, p , selects the output value x . All the CLB outputs can be either direct, inverted or registered.

Note that one XC4000 CLB, although it can accept 9 distinct inputs, is not equivalent to a 9-input LUT. A 9-input LUT can implement any functions of 9 distinct variables while one XC4000's CLB can implement any functions of 5 distinct inputs and some functions of 6 to 9 variables. For example, expressions in both (a) and (b) in Figure 3 have 9 literals, but (a) needs 1 CLB whereas (b) needs 2 CLBs.

2.2 XC4000 Programmable Interconnect Point and Routing Resources

Xilinx XC4000 routing resources are connected by switch matrices. There are 8 (6 for smaller devices) intersections containing 6 programmable interconnect points (PIPs) each. The PIP, shown schematically in Figure 4(c), is a pass transistor controlled by a configuration memory cell.

XC4000 routing resources include *single-length (general-purpose) lines* (SLs), shown in Figure 4(a), *double-length lines* (DLs), shown in Figure 4(b) and *long lines* (LLs). LLs run

Figure 3: One XC4000 CLB can not implement all the functions with less than 10 distinct inputs (a) 9 literals, 9 distinct inputs need 1 CLB (b) 9 literals, 6 distinct inputs need 2 CLBs.

Figure 4: Wiring architecture (a) single length lines (b) double length lines (c) the XC4000 switch metric, connections and PIP.

the width or height of the chip with negligible delay variations. SLs connect every pair of adjacent switch matrices ¹ and DLs by-pass alternate switch boxes ². Thus, the wirability of a net is no longer a simple function of its length and the congestion of its routing region. On the other hand, since signal delay depends more on the number of PIPs through which a signal passes than on the length of the segments, the double-length lines allow a signal to travel twice the distance in the same amount of time, or to travel a same distance in half the time as the single length lines do ³. The delay of a wire is also no longer a simple function of its length.

3 Previous Work

Several fast mapping heuristics for LUT based FPGAs are surveyed in [3]. Such heuristics can be used to obtain estimation of CLB count. However, techniques for timing estimation haven't been proposed so far.

Xilinx's [4] Partitioning, Placement and Routing (PPR) software package has its own built-in estimation tool. This estimation is very accurate since it performs the actual mapping using Chortle [5], but the tool does not provide performance estimation.

Other than Xilinx, Synopsys [6] also provides accurate area estimation by doing actual mapping. Moreover, it can provide estimation of the number of logic levels for the design. Nevertheless, it doesn't take into account wiring delay.

The research presented in [7] empirically examines the performance of multi-level logic minimization tools for a LUT based FPGA technology and suggests that there is a linear relationship between the number of literals and the number of routed CLBs. It provides estimation for both area and timing but the work is only applicable to the XC3000 series.

On the other hand, our area and timing estimation for LUT based FPGA concentrates on Xilinx XC4000. It has a more realistic and accurate model than others proposed in the past, since it takes into account not only the gate area/delay but also the wiring effects. Additionally, our approach is easy to adapt to other Xilinx series such as XC2000 and XC3000 with minor modifications.

¹The wire between two adjacent switch matrices is a SL segment.

²The wire connect every other switch matrices is a DL segment.

³Experiments show that SL segments and DL segments have approximately the same delay.

4 Estimation for LUT Based FPGAs

4.1 Problem Definition

Following Xilinx's standard methodology, we assume that an RT level design generated by HLS has been translated into a logic netlist⁴. Given a structural logic level netlist description, the goal of FPGA area estimation is to predict the area of the design in terms of the number of CLBs. Afterwards, FPGA timing estimator accepts the netlist with placement information predicted by the area estimation predicts the timing. Furthermore, it is crucial that both estimation schemes be fast and accurate.

4.2 Deficiencies of Standard Area Estimation Models

In an FPGA, the size of logic is reflected by the number of LUTs, and subsequently the number CLBs. Over the years, gate count has been used extensively in estimating logic size. However, it can not be used in estimating the number of LUTs [8]. For example, consider a 3-input LUT⁵ in Figure 5. The mapping shown in Figure 5(a) has 8 gates and needs 1 LUT while the mapping shown in Figure 5(b) has only three gates but needs 2 LUTs.

Figure 5: Gate/input count is not a good measure of LUT count (a) 8 gates, 3 inputs need 1 LUT (b) 3 gates, 4 inputs need 2 LUTs (c) 6 gates, 4 inputs need 3 LUTs.

Moreover, while the number of inputs of an LUT is limited, distinct input count is not a good measure of LUT count either. For example, in Figure 5, the mappings shown in both

⁴Our estimation framework also supports a true hierarchical design approach [2].

⁵We use 3-input LUT to make the example easier to understand.

(b) and (c) have four distinct inputs each, but mapping in (b) needs 2 LUTs and mapping in (c) needs 3 LUTs.

Although one Xilinx XC4000 CLB can contain three LUTs, the number of CLB is not constrained by the number of LUTs. For example, in Figure 6, the mappings shown in (a) and (b) have 3 LUTs but the mapping in (a) needs 1 CLB and the mapping in (b) needs 2 CLBs.

Figure 6: Number of CLB is not constrained by number of LUT (a) 3 LUTs need 1 CLB (b) 3 LUTs need 2 CLB.

Figure 7: Net length does not necessarily correlate well with performance (a) same Manhattan distance connections with different delays (b) different Manhattan distance connections with same delays.

4.3 Wiring Delay in FPGAs

In Xilinx architecture, there are three types of wiring, i.e. single-length lines, double-length lines, and long lines. Thus, the delay of a net is mainly decided by the number of segments and the number of PIPs it goes through. As a result, net length does not necessarily correlate well with either routability or performance. For example, Figure 7(a) shows that nets with the same distance may have different delays, while Figure 7(b) shows that nets with different distances may have the same delays. Because of this, standard delay models cannot be directly applied to FPGAs.

4.4 Area Estimation Approach

Figure 8: Area estimation approach.

Our area estimation approach consists of: (1) predicting LUT mapping, (2) predicting CLB construction and (3) predicting the placement and the shape. It is shown in Figure 8. We shall use the walk-through example shown in Figure 9 to explain our approach.

4.4.1 Predicting LUT Mapping

To predict LUT mapping, the algorithm first decomposes the input netlist into a fanout-free tree forest as shown in Figure 9(a) ⁶. Then, the netlist is preprocessed by temporarily

⁶While this decomposition may not result in a mapping with a minimal number of LUT, it nevertheless

Figure 9: A walk-through example (a) predicting LUT mapping (b) predicting CLB construction (c) predicting placement and shape.

discarding inverters and flip-flops since CLBs have built-in inverters and flip-flops at all their outputs. Then, starting from the output of each tree, the algorithm does a depth-first traversal and groups the gates in the netlist into K -input LUTs. To generate K -input LUT, the algorithm checks whether each visited gate can be included into the current LUT. If the total number of distinct inputs thus far is less than $K + 1$, then it is included, otherwise, it is rejected from inclusion in the current LUT.

Because of the special XC4000 architecture, there are several possible CLB configurations. CLBs with different configurations have different LUT patterns and delays. Figure 10 shows three possible CLB configurations.

There are three different LUT patterns in XC4000 CLB. They are F-G-H pattern shown in Figure 10(a), F-H pattern shown in Figure 10(b) and F/G pattern shown in Figure 10(c). When we do the grouping, the algorithm tries to follow these patterns. For example, to construct a F-G-H pattern, the algorithm first construct a H-LUT, then tries to construct one F-LUT and one G-LUT. Priority is given to all these patterns with the priority order shown in Figure 10. In Figure 10, both patterns shown in (a) and (b) can make good use of the H LUT and therefore would result in a more compact CLB netlist. We give the pattern shown in Figure 10(c) the lowest priority since it leaves the H LUT unused. Another reason for this priority ordering has to do with delay. Both patterns shown in (a) and (b) use the H LUT as a second logic level instead of another F or G LUT, thus resulting in smaller input-to-output delay. As a result, the overall circuit will be faster.

The detailed LUT mapping algorithm is shown if Figure 11 and Figure 12. Since the makes the mapping prediction algorithm faster and more efficient.

Figure 10: Different CLB configurations.

input netlist is decomposed into a fanout-free tree forest at the beginning, the LUT mapping algorithm will not traverse any gate more than twice (one for rejecting it from inclusion in a LUT, another one for accepting it), therefore, the time complexity of the algorithm is $O(n)$ where n is the number of 2-input gates.

4.4.2 Predicting CLB Construction

Starting with the LUT netlist, the CLB construction algorithm tries to group LUTs into CLBs. In Figure 9(b), with the information obtained during LUT mapping, we first construct F-G-H pattern CLB, CLB_A. Then, we construct F-H pattern and find another G-LUT, which has the most common inputs with the current F-H pattern, and pack them into one CLB, CLB_C; Finally, try to pack two LUTs into one CLB. By trying to pack LUTs with the most common inputs into one CLB, the number of interconnections between CLBs can be minimized.

The detailed CLB generate algorithm is shown in Figure 13. Since we visit every LUT only once, the time complexity of the algorithm is $O(n)$, where n is the number of LUTs in the LUT netlist. At this point, not only the number of CLBs, but also an actual CLB netlist is obtained.

4.4.3 Predicting Placement and Shape

Because of the granularity of FPGAs where area is in terms of CLBs rather than in terms of square microns in custom design, reducing unused area is more important than ever. To

Algorithm lookup_table_mapping()

Inputs: Structural gate level netlist for components .

Output: Lookup-table netlist

begin Algorithm

```
while (list_of_outputs  $\neq$  {}){
  select an output (O) from list_of_outputs;
  switch(np(O)){ /* np(O) = number of predecessors of O which are not primary inputs */
    case 1: np(O) == 2
      generate_lut_4(lprd(O)); /* lprd = left predecessor */
      generate_lut_4(rprd(O)); /* rprd = right predecessor */
      merge_lut(O, lprd(O), rprd(O)); /* try to merge F-G-H MAP into F or F-H MAP */
      update_list_of_outputs; /* add non-primary lookup-tables to list_of_outputs */
      break;
    case 2: np(O) == 1
      p = pred(O);
      switch(np(p)){
        case 1: np(p) == 2
          generate_lut_4(lprd(p));
          generate_lut_4(rprd(p));
          merge_lut(O, p, lprd(p), rprd(p));
          update_list_of_outputs;
          break;
        case 2: np(p) == 1
          generate_lut_4(prd(p)); /* prd = predecessor of p */
          merge_lut(O, p, prd(p));
          update_list_of_outputs;
          break;
        case 3: np(p) == 0
          merge_lut(O, p);
          update_list_of_output;
          break;
      }
      break;
    case 3: np(O) == 0
      merge_lut(O);
      update_list_of_outputs;
      break;
  }
}
```

end Algorithm

Figure 11: Lookup table mapping algorithm.

Algorithm generate_lut_4(gate_index)

Inputs: Index of one gate

Output: One 4-inputs lookup table start from the gate given by gate_index;

begin Algorithm

```
depth_first_traverse(gate_index) the net list;  
if(num_distinct_inputs  $\leq$  5)then  
    add_gate_to_lut(gate_index);  
    depth_first_traverse(prd);/*prd = predecessor of the gate specified by gate_index */  
else return;  
endif
```

end Algorithm

Figure 12: Lookup table generate algorithm.

achieve this goal, first we enumerate all possible *aspect ratios* (width vs. height) based on the number of CLBs in the netlist produced. Then for each aspect ratio, we build a *slicing tree* by recursively slicing a design down to individual CLBs based on min-cut partitioning. For example, in Figure 14, the component consists 9 CLBs. If we consider aspect ratio equals to 1, we get slicing tree (a); if we consider aspect ratio equals to 2.5, we get another slicing tree shown in (b); The partitioning cut line is always placed halfway through the width of the current slice when the width is larger than the height, otherwise, it is placed halfway through the height of the current slice. Figure 9(c) shows a slicing tree and its corresponding placement for our walk-through example with an aspect ratio of 1. After predicting the placement and the shape, we obtain a complete slicing tree for the CLB netlist.

Although the slicing tree construction is aspect ratio dependent, this constructive approach does not consume excessive run time since the number of CLBs are limited to a relatively small number: from 8x8 to 24x24⁷.

When we build the slicing tree, the wiring area is also taken into account by adjusting the shape of the slices. The adjustment is done by comparing the interconnections between every two sibling blocks in the slicing tree with the available routing resource budget. The amount of routing needed for connecting the two blocks can be obtained by estimating the interconnection count between them.

The available routing resource budget will depends on the shapes and sizes of the two sibling blocks as shown in Figure 15. In the intervening routing channel between the two sibling blocks, there are six single-length lines between every pair of adjacent switch matrices that are parallel to the slice orientation. In addition, we assume that double-length lines

⁷In general, most designs only occupy up to half or one third of the entire FPGA.

Algorithm clb_generate()

Inputs: Lookup table netlist .

Output: CLB netlist

begin Algorithm

```
decompose graph into fanout free tree forest;
while (list_of_H - MAP  $\neq$  {}){
  select an H-MAP (O) from list_of_H-MAP;
  switch(np(O)){ /* np(O) = number of predecessors of O which are not primary inputs */
    case 1: np(O) == 2
      generate_clb(O,lprd(O),rprd(O)) ; /* group O and it's two predecessor into one CLB */
      /* lprd(O) = left predecessor, rprd(O) = right predecessor */
      update_list_of_F-MAP(lprd(O), rprd(O)); /* delete lprd(O), rprd(O) from list_of_F-MAP */
      /* lprd(O)=left predecessor , rprd(O)=right predecessor */
      update_list_of_H-MAP(lprd(O), rprd(O));
      /* delete lprd(O), rprd(O), O from list_of_H-MAP */
      break;
    case 2: np(O) == 1
      fm=find_match_lut(O, prd(O)); /* prd(O) = predecessor */
      /* find lut which shares most inputs with O and prd(O) */
      generate_clb(O, prd(O), fm);
      /* group O, prd(O) into F-H map and compact it together with fm into one CLB */
      update_list_of_F-MAP(prd(O)); /* delete prd(O), fm from list_of_F-MAP */
      update_list_of_H-MAP(O, prd(O), fm); /* delete O, prd(O), fm, from list_of_H-MAP */
      break;
    case 3: np(O) == 0
      put (O) into list_of_F-MAP
      update_list_of_H-MAP(O); /* delete O from list_of_H-MAP */
      break;
  }
}
while (list_of_F - MAP  $\neq$  {}){
  select an F-MAP (O) from list_of_F-MAP;
  fm=find_match_lut(O); /* find lut which shares most inputs with O */
  generate_lut(O, fm); /* patch O and fm into one CLB */
  update_list_of_F-MAP(O, fm); /* delete O, fm from list_of_F-MAP */
}
```

end Algorithm

Figure 13: CLB generate algorithm.

Figure 14: An Example with 9 CLBs (a) slicing tree and the placement($asp = 1$) (b) slicing tree and the the placement ($asp = 2.5$).

perpendicular to the slice orientation are also used in that channel, while the ones parallel to the slicing orientations are reserved for the parent level in the slicing tree. Thus, the total available routing budget can be calculated based on the size of the slicing cut (i.e. the length of the routing channel) between the two sibling blocks as following.

Let W_p be the width of the current parent slice, H_p be the height of the current parent slice, the total budget for interconnecting two horizontal slicing child-slices can be calculated as follows: (where α is the single lines count: 6 for bigger devices and 4 for smaller devices).

$$\begin{aligned} total_routing_budget &= \alpha * W_p + 2 * (W_p + 1) \\ &= (\alpha + 2) * W_p + 2; \end{aligned}$$

Similarly, from Figure 15(b), the total budget for interconnecting two vertical slicing child-slices can be calculated as follows:

$$\begin{aligned} total_routing_budget &= \alpha * H_p + 2 * (H_p + 1) \\ &= (\alpha + 2) * H_p + 2; \end{aligned}$$

Figure 15: Wire budget (a) with horizontal slice line (b) with vertical slice line.

When the required routing resources exceed the budget, the adjustment is needed so as to accommodate the extra routing requirements. Again, let num_con be the expected number of interconnection between two child slices, we can estimate the dilation area by using the following equations.

For a vertical slice,

$$\Delta W_p = \text{Max}\{0, (num_con - total_routing_budget) / 2 * H_p * \alpha\};$$

Similarly, for a horizontal slice,

$$\Delta H_p = \text{Max}\{0, (num_con - total_routing_budget) / 2 * W_p * \alpha\};$$

The adjustment area can then be added to the current composite block which is formed by combining the two sibling blocks. Additionally, all the parent blocks in the slicing tree are correspondingly adjusted as well. At this moment, we have an approximate topology of the design which can be used in the timing models described in next section. Figure 16 shows a typical approximate topology for one benchmark example: apex7 [10].

4.5 Timing Estimation Approach

In an FPGA, there are block delays, wire segment delays and PIP delays. Our timing estimation model includes CLB delays, and wiring delays, and is used to predict the input-to-output worst case delay, both before and after timing optimization. It is shown in Figure 17.

C35	C3	C15	C9	C33	C31	C25	C4
C2	C37	C10	C46	C1	C59	C5	C32
	C26		C16	C23	C11		C24
C34	C49	C18	C45	C22	C50	C6	C29
C42	C38	C48	C12	C55	C30	C53	
C20	C47	C40	C41	C14	C58	C13	C57
C52	C8		C19	C54	C28	C51	C21
C39	C56	C17	C44	C27	C43	C7	C36

Figure 16: Approximate topology for apex7.

Figure 17: Timing estimation approach.

4.5.1 Predict CLB Delay

As we mentioned in Section 4.4.1, we know that XC4000 CLB has different configurations. For different CLB configurations, the outputs from the CLBs have different delays. The delay for each of those configurations can be obtained from Xilinx's data book [4] and is shown in Figure 18.

parameters speed grade	dtv/dgh	dt/dg	dh	dtg/dgt
4	6.0	4.0	4.5	8.0
5	7.0	4.5	5.0	9.0
6	8.0	6.0	7.0	12.0

Figure 18: The parameters' value for different kind of CLB outputs.

4.5.2 Predict Wiring Delay

To predict the delay between point A and B, $D(A, B)$ in Figure 19, the Mahanttan distance x, y are first calculated using following formulas:

$$x = |A_x - B_x|$$

and

$$y = |A_y - B_y|$$

Then, wire type (single-length lines, double-length lines and long lines) is assigned to the wire. This also decides the number of PIPs and number of segments between points A and B, and subsequently, the point-to-point delay without fanout effects, $D_{pp}(A, B)$, can then be calculated. Finally, the delay with fanout effects, $D(A, B)$, can be obtained by adjusting the non-fanout point-to-point delay with fanout factor.

To predict the wire type, the algorithm mainly checks the interconnect wire length x and y respectively. First, the long lines are assigned for all the wires which are longer than 8 CLBs. Then, single-length lines are assigned for all wires which are shorter than 2 CLBs. Since single-length line can not be connected with double-length lines, if the length of the wire connected with single-length line is between 2 and 8 CLBs, a single-length line

point-to-point delay:

for single/double length lines —

$$D_{pp}(A, B) = \#PIP \times d_{pip} + \#seg \times d_{seg}$$

for long lines —

$$D_{pp}(A, B) = d_{||}(x + y)$$

Figure 19: Point-to-point delay model and associated parameters.

is also assigned to that wire. Finally, double-length lines are assigned for the rest of the interconnect wires. The detailed algorithm is shown in Figure 20.

From Section 4.3, we know that net length does not necessarily correlate well with the actual delay. Therefore, we use an empirical model to characterize the delay-vs-wiring-type relationship. Our empirical model is based on a large number of observations obtained by using Xilinx's XDM layout tool to place and route a set of benchmarks and analyzing the delay of each point-to-point connection using *Xdelay*, the Xilinx timing analysis tool. We found that it is satisfactory to approximate the delay as a function of (1) the number of PIPs it goes through in both X and Y directions respectively, and (2) the corresponding segment delays. Let's denote the delay for each PIP in the programmable switch matrices as d_{pip} , and the delay for each segment as d_{seg} . Note that we use the same variable d_{seg} for both single-length and double-length segments since experiments show that their delays are approximately the same⁹. For a 2-point net (A, B), the point-to-point delay will be the summation of such delays in both X and Y directions. Let x and y be the Mahanttan distances of (A, B) in X and Y direction respectively. If only single-length lines are used, they will pass through x and y PIPs, and through $x + 1$ and $y + 1$ segments in the X and Y directions respectively. Double-length lines need one PIP in every other CLB and, similarly, for segments on same distance as single-length line interconnection. Long lines with same length will not go through any PIPs and eventually the long line delay is approximated as being proportional to the wire length. Thus, the point-to-point delay will be:

⁹The model can be easily modified to account for different delays of single-length and double-length segments, if needed.

Algorithm routing(x, y)

Inputs: Distance between one pair of source-sink pins in the approximate topologies: x, y .
Output: $num_pips_x, num_pips_y, num_seg_x, num_seg_y, total_num_pips$

begin Algorithm

```
if ( $x \geq 8(CLBs)$ ) then  $x$  use long line;
if ( $y \geq 8(CLBs)$ ) then  $y$  use long line;
if ( $x < 2(CLBs)$ ) then  $x$  use single-length line;
     $num\_pips_x = x$ ;
     $num\_seg_x = num\_pips_x + 1$ ;
    if ( $y < 8(CLBs)$ ) then  $y$  use single-length line;
         $num\_pips_y = y$ ;
         $num\_seg_y = num\_pips_y + 1$ ;
    end if;
end if;
if ( $y < 2(CLBs)$ ) then  $y$  use single-length line;
     $num\_pips_y = y$ ;
     $num\_seg_y = num\_pips_y + 1$ ;
    if ( $x < 8(CLBs)$ ) then  $x$  use single-length line;
         $num\_pips_x = x$ ;
         $num\_seg_x = num\_pips_x + 1$ ;
    end if;
end if;
for the rest of  $x$  and  $y$  use double-length lines;
     $num\_pips_x = x/2$ ;
     $num\_seg_x = num\_pips_x + 1$ ;
     $num\_pips_y = y/2$ ;
     $num\_seg_y = num\_pips_y + 1$ ;
end for;
```

end Algorithm

Figure 20: Routing rule algorithm.

Algorithm point-to-point delay(x, y)

Inputs: $x, y, num_pips_x, num_pips_y, num_seg_x, num_seg_y$.
Output: point to point delay: $delay_pp$

begin Algorithm

```

if (( $x \geq 8(CLBs)$ ) AND ( $y \geq 8(CLBs)$ )) then /*assume long line*/
     $delay\_pp = \alpha + p_{ll} * (x + y + 1)$ ;
end if;
if (( $x \geq 8(CLBs)$ ) AND ( $y < 8(CLBs)$ )) then /* assume  $x$  use long line */
     $delay\_pp = 2 * \alpha + p_{ll} * x + d_{seg} * num\_seg_y + d_{pip} * num\_pips_y$ ;
end if;
if (( $y \geq 8(CLBs)$ ) AND ( $x < 8(CLBs)$ )) then /* assume  $y$  use long line */
     $delay\_pp = 2 * \alpha + p_{ll} * y + d_{seg} * num\_seg_x + d_{pip} * num\_pips_x$ ;
end if;
if (( $x < 8(CLBs)$ ) AND ( $y < 8(CLBs)$ )) then /*assume no long lines are used*/
    if(( $num\_pips_x == 0$ )AND( $num\_pips_y == 0$ )) then
        /* assume direct interconnect without going through pips */
         $delay\_pp = \beta$ ;
    else /* assume single/double-length interconnection */
         $delay\_pp = \alpha + d_{pip} * (num\_pips_x + num\_pips_y) + d_{seg} * (num\_seg_y + num\_seg_x)$ ;
    end if;
end if;

```

end Algorithm

Figure 21: Delay Algorithm.

ϵ	d_{pip}	d_{seg}	d_{ll}	α	β
4	0.3	0.3	0.18	0.6	1.3

Figure 22: The parameters for the delay model and timing optimization.

$$D_{pp}(A, B) = \begin{cases} d_{pip} * x + d_{seg} * (x + 1) + d_{pip} * y + d_{seg} * (y + 1) & \text{for single length lines} \\ d_{pip} * \frac{x}{2} + d_{seg} * (\frac{x}{2} + 1) + d_{pip} * \frac{y}{2} + d_{seg} * (\frac{y}{2} + 1) & \text{for double length lines} \\ d_{ll} * (x + y) & \text{for long lines} \end{cases}$$

and the detailed algorithm is shown in Figure 21 and the associated parameters are listed in Figure 22.

When the number of fanout of a net is larger than one, say f , the delay on each sink pin j ($j = 1, \dots, f$) will be affected by the delay on the rest of sink pins k ($k = 1, \dots, f; k \neq j$) on the net. Let i be the source pin, for each sink pin j ($j = 1, \dots, f$). The point-to-point delay without fanout effect $D_{pp}(i, j)$ is first computed. Afterwards, we denote $D(i, j)$ as the delay with fanout effects, and it can be obtained by adjusting the point-to-point delay without

fanout effects, $D_{pp}(i, j)$, using the following formula:

$$D(i, j) = D_{pp}(i, j) + \frac{1}{\epsilon} \sum_{k=1, \dots, f; k \neq j} D_{pp}(i, k)$$

Where, ϵ , a fanout adjustment factor, is experimentally obtained as 4. At the end of this step, we have a netlist which contains estimates of CLBs and net delays.

4.5.3 Predict Input-to-output Worst Case Delay

The timing estimation algorithm computes the delay between an input and an output port of a combinational circuit consisting of logic gates (or flip-flops of a sequential circuit). The circuit is traversed from the input to the output ports. The CLB delays and the wire delays are used to calculate the delay of each path. The critical path is the path that yield the longest input-to-output delays. A more detailed description can be found in [9].

4.5.4 Predict Timing Optimization

In Xilinx, after performing partitioning, placement and routing, the performance is further optimized by re-routing nets on the critical path. The critical path can be obtained by running Xilinx PPR followed by static timing analysis. This optimization would assign the nets on the critical path to long lines as much as possible. It usually reduces the PIPs as well as segment delays on these nets. Our timing estimator predicts the timing optimization by analyzing the delay of the nets on the critical path and estimating the effects of corresponding modifications.

Assume that a fraction, f_{sl} , of the nets on the critical path are single-length lines, and a fraction, f_{dl} , of the nets are double-length lines. Timing optimization will substitute both of them with long lines. Let $D(n)$ be the original net delay, and $D(n)^*$ be the delay after timing optimization, we define the delay reduction factor μ_{opt} as follows:

$$\mu_{opt} = \frac{D(n)^*}{D(n)} = \frac{D_{pp}(ll)}{f_{dl} * d_{pp}(dl) + f_{sl} * d_{pp}(sl)}$$

On an average, we expect single-length lines and double-length lines to be equally used on the critical path. In the worst case, no nets on the critical path has been assigned to long lines before timing optimization, thus $f_{sl} = f_{dl} = 0.5$. Substituting into the equation above, we get $\mu_{opt} = 0.4$ for the parameter values shown in Figure 19. Our timing estimator

predicts the effects of timing optimization by reducing the wiring delay on critical paths with the timing reduction factor μ_{opt} , and hence obtain the delay after timing optimization with almost no extra run time.

5 Experimental Results

5.1 Experimental Procedure

We have implemented our area and timing estimator in C on Sun SPARC workstation. The designs used to test our estimator are from the standard logic synthesis benchmark set [10]. Given a general behavioral description, MIS [11] is first used to do logic optimization and synthesis in order to get a structural netlist of 2-input gates. Then, Xilinx's PPR is invoked to generate the layout and Xdelay to obtain timing information of each circuit. We accept the same VHDL structural netlist as does PPR, and invoke the area estimator to obtain the area estimates. Then our timing estimator accepts the output from the area estimator and produces timing estimates before and after timing optimization. Because of the non-deterministic nature of PPR, the designer tends to run PPR several times with different seeds and selects the best one (in the experiments we conducted, the worst case delay varied from 4.4% to 20.9% off the best case in ten PPR runs). Because of this, PPR and Xdelay were run 10 times with different seeds and the best design was selected to run Xilinx's timing optimization program. Then, Xdelay is used to get the measured delay information from the physical design.

5.2 Results

Figure 23(a) shows the comparison between measured results and our estimation results for all the benchmarks. First, note that our area estimator accurately predicted the shape of the design. Our estimator can produce accurate estimates within a matter of seconds. The average estimation error in delay is about 5.6%, while the worst case error is 13.2%. Even for one run of PPR/Xdelay, our estimator is still at least an order of magnitude faster than PPR/Xdelay.

Figure 23(b) shows the comparison between measured results after timing optimization and our estimation results. We get this estimation with almost no extra run time. The average estimation error in timing is about 6.1% while the worst case error is 15.8%. The Xilinx's run time shown in Figure 23(b) includes one PPR run with timing optimization

Benchmark design	Number of gates	Measured area(CLB)	Estimated area(CLB)	% error area	Measured delay(ns)	Estimated delay(ns)	% error delay	Xilinx run time (s)	Estimation run time (s)
z4mi	202	40 (7x7)	44 (7x7)	0	39.9	41.7	+4.5	900	5
vg2	645	132 (12x12)	133 (12x12)	0	59.2	67.0	+13.2	2930	29
sy9mmi	223	42 (7x7)	46 (7x7)	0	54.0	55.1	+2.0	810	5
sy9m	369	81 (9x9)	90 (9x10)	+11.1	74.1	75.7	+2.2	1610	12
pp1	240	40 (6x7)	42 (6x7)	0	36.0	34.6	-3.9	910	6
duke2	617	135 (12x12)	136 (12x12)	0	82.4	74.5	-9.6	2570	31
apex7	321	58 (8x8)	59 (8x8)	0	69.3	70.9	+2.3	1520	11
C499	495	81 (9x9)	60 (8x8)	-11.1	83.3	88.8	+6.6	1280	21
C880	394	76 (9x9)	76 (9x9)	0	110.6	117.2	+6.0	1440	17
Average error				2.5			5.6		

(a)

Benchmark design	Measured delay(ns)	Estimated delay(ns)	% error (delay)	Xilinx run time (s)	Estimation run time (s)
z4mi	36.5	36.2	-0.8	209	5
vg2	47.6	54.9	+15.3	704	29
sy9mmi	46.4	46.8	+0.9	162	5
sy9m	58.2	57.2	-1.7	333	12
pp1	30.8	30.7	-0.3	174	6
duke2	53.2	53.7	+0.9	844	31
apex7	51.3	59.4	+15.8	325	11
C499	82.9	70.6	-14.8	348	21
C880	88.1	91.8	+4.2	372	17
Average error			6.1		

(b)

Figure 23: Experiment results (a) unoptimized results (b) optimized results.

activated and Xdelay run time. Note that timing optimization is a very time consuming process, because of the possible re-routing caused by the optimization. These results clearly indicate that our tool can be efficiently used to provide fast and accurate feedback to synthesis tools, allowing them to make better informed design decisions.

6 Conclusion

We presented an area and timing estimation for LUT based FPGAs approach based on a realistic and accurate model which takes into account not only gate area and delay but also wiring effects. The estimation is accurate and fast, as supported by standard benchmarks, on which the estimates were in error by an average of 6.1% and took a matter of seconds to compute. Future work will concentrate on linking estimation to synthesis.

7 References

- [1] D.D. Gajski, L. Ramachandran, P. Fung, S. Narayan and F. Vahid, "100-hour Design Cycle: A Test Case," *Proc. Euro DAC*, 1994
- [2] M. Xu, F.J. Kurdahi, "Chip Level Area and Timing Estimation for Lookup Table Based FPGAs," *Technical Report #95-31, UCI*, Aug.1995.

- [3] Robert J. Francis, "A Tutorial on Logic Synthesis for Lookup-Table Based FPGAs," *Proc. ICCAD*, 1992.
- [4] Xilinx, "XACT Development System: Libraries Guide," *Xilinx*, 1994.
- [5] R.J. Francis, J. Rose, Z. Vranesic, "Chortle: A Technology Mapping Program for Lookup Table-Based Field-Programmable Gate Arrays," *Proc. 27th DAC*, June 1990.
- [6] Xilinx, "XACT Xilinx Synopsys Interface FPGA User Guide," *Xilinx*, 1995.
- [7] M.D.F. Schlag, P.K. Chan, and J. Kong, "Empirical Evaluation of multilevel Logic Minimization Tools for a Field-Programmable Gate Array Technology", *Technical Report. University of California, Santa Cruz*, 1991.
- [8] S. Trimberger, "Field Programmable Gate Array Technology," *Kluwer Academic Publishers*, 1994.
- [9] C. Ramachandran and F.J. Kurdahi, "Combined Topological and Functionality-Based Delay Estimation Using a Layout-Driven Approach for High-Level Applications", *IEEE Transaction on CAD*, pp.1450-1460, Dec.1994.
- [10] A.J. de Geus, "Logic Synthesis and Optimization Benchmarks for the 1986 Design Automation Conference" *Proc. of the 23rd Design Automation Conference*, June 86, 78.
- [11] R.K. Brayton, R. Rudell, and A.L. Sangiovanni-Vincentelli, "MIS: A multiple-level logic optimization," *IEEE Transaction on CAD*, pp. 1062-1081, Nov. 1987.