

ChipEst-FPGA: A Tool for Chip Level Area and Timing Estimation of Lookup Table Based FPGAs for High Level Applications

Min Xu

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, U.S.A.
Tel: 714-824-8168, Fax: 714-824-4056
e-mail: mxu@ics.uci.edu

Fadi J. Kurdahi

Dept. of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92697-3425, U.S.A.
Tel: 714-824-8104, Fax: 714-824-2321
e-mail: kurdahi@ece.uci.edu

Abstract

The importance of efficient area and timing estimation techniques for hierarchical design methodology is well-established in High-Level Synthesis (HLS), since the estimation allows more realistic exploration of the design space, and hierarchical design methodology matches well with HLS paradigm. In this paper, we present ChipEst-FPGA, a chip level estimator for designs implemented using a hierarchical design methodology for Lookup Table Based FPGAs. In FPGAs, the wire delay may contribute to a significant portion of the overall design delay. ChipEst-FPGA uses a realistic model which takes the component area/delay as well as wiring effects into account. We tested our ChipEst-FPGA on several benchmarks and the results show that we can get accurate area and timing estimates efficiently.

1 Introduction

The ability to shorten development cycles has made Field Programmable Gate Arrays (FPGAs) 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 starting from an idea to a final product.

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). **Partitioning** (or **technology mapping**), **placement**, **routing** and **timing optimization**. This is a flat design approach since the netlist fed into partitioning is a gate level netlist and the partitioning is done on the whole netlist (more detailed discussion can be found in [9]).

Contrast to this flat design flow, Figure 1(b) shows a hierarchical HLS design flow targeted for FPGAs. It has an **RT level technology mapping** step which partitions the incoming netlist into RT level components¹ and maps them onto pre-characterized components or uses layout tools do the components layout. This way, the structural information is preserved in each component.

¹we refer to individual registers, counters, adders, muxes, RAM arrays etc as RT level components.

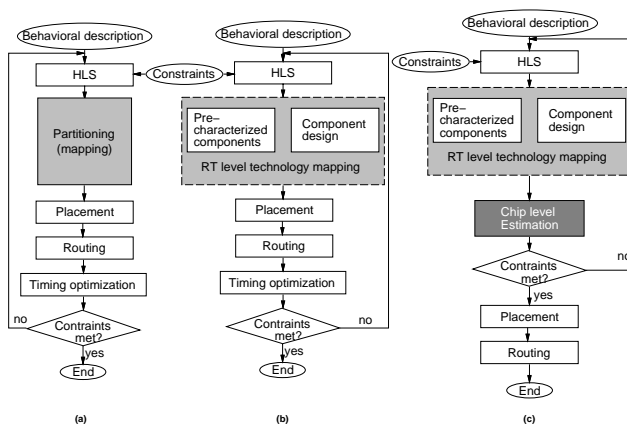


Figure 1: Flat HLS design flow vs. hierarchical HLS design flow targeted for FPGAs: (a) flat design flow (b) hierarchical design flow (c) the importance of estimation in a typical hierarchical HLS design flow.

Maintaining this hierarchy is beneficial because of the following reasons. (1) It is easy to do debug, easy to add or change logic since design changes in one component can be made without affecting the placement and routing of the rest of the design. (2) It is easy to adapt to different technology. (3) It is easy to improve the design routability by grouping and floorplanning the RT components according to the data flow. It is easy to improve the design's performance. (4) It matches well with the HLS design paradigm since the hierarchy is maintained through out the design process. (5) With proper binding and component selection, it is possible to optimize the overall design by selecting different component implementations for different datapath operations. Thus, multiplications by constants can be replaced by simpler components. This offsets potential shortcomings of simpler HLS-based RT level design paradigms which assume one implementation style per component.

In the hierarchical HLS design flow targeted for FPGAs, placement and routing 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 to a significant portion of the overall design delay. The problem becomes especially acute when the design process starts at the be-

havioral 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 the time consuming placement and routing phases as shown in Figure 1 (c). It is very important that the estimator has a more realistic and accurate model which takes into account not only component area and timing, but also wiring effects.

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. To be specific, we target the Xilinx XC4000 series because of their popularity. In addition, our tools can be used for designing large systems which span several FPGAs: By coupling a high level synthesis tool with our estimation tools, it is possible to explore a large number of system level partitioning alternatives, either interactively or automatically. An example of such a paradigm is described in the Spec-Syn system [12]. Finally, our tools can also be used for manually generated RT level designs to provide an almost instantaneous feedback to the designer on the quality of a particular implementation.

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). In later versions of their CAD tools, Xilinx appears to be moving towards promoting hierarchical design flow by introducing **Hard Macros**, **hmgen**, and **RPMs** [10]. By using Hard Macros or RPMs, the component placement information is preserved in hard macros.

2.1 XC4000 Configurable Logic Blocks and Lookup Tables

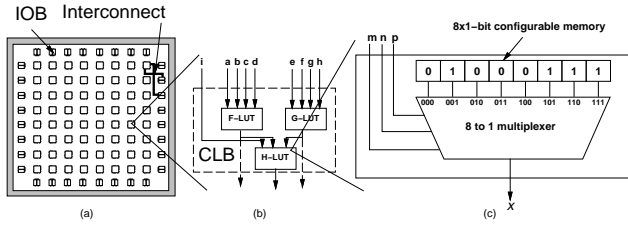


Figure 2: Xilinx XC4000 architecture (a) XC4000 (b) abbreviated CLB architecture (c) a LUT implementing $x = mn + np$.

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. All the CLB outputs can be either direct, inverted or registered.

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 3(c), is a pass transistor controlled by a configuration memory cell.

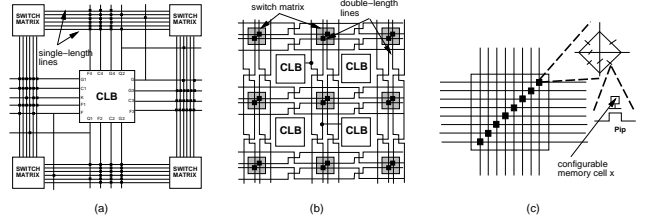


Figure 3: Wiring architecture (a) single length lines (b) double length lines (c) the XC4000 switch metric, connections and PIP.

XC4000 routing resources include *single-length (general-purpose) lines* (SLs), shown in Figure 3(a), *double-length lines* (DLs), shown in Figure 3(b) and *long lines* (LLs). LLs run the width or the height of the chip with negligible delay variations. SLs connect every pair of adjacent switch matrices² and DLs by-pass alternate switch boxes³. 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 do⁴. The delay of a wire is also no longer a simple function of its length. Nets with the same distance may have different delays while nets with different distances may have the same delays [9].

3 Previous Work

Several fast mapping heuristics for LUT based FPGAs are surveyed in [6]. Such heuristics can be used to obtain estimation of CLB count. However, techniques for timing estimation haven't been proposed so far.

Xilinx's [3] 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 [4], but the tool does not provide performance estimation.

Other than Xilinx, Synopsys [5] 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 [2] empirically examines the performance of multi-level logic minimization tools for a

²The wire between two adjacent switch matrices is a SL segment.

³The wire connect every other switch matrices is a DL segment.

⁴Experiments show that SL segments and DL segments have approximately the same delay.

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.

CompEst-FPGA [9] presented an area and timing estimation for LUT based FPGAs approach. It takes into account gate area/delay as well as wiring effects. It can handle Xilinx XC4000 estimation.

All those approaches are suitable to estimate component level design and chip level design but with flat design methodology. None of them supports a hierarchical design methodology.

The work presented here is the extension of our work presented in [9]. It has a realistic and accurate model since it takes into account not only the component area/delay but also the wiring effects. It mainly handles hierarchical design methodology for high level applications. Additionally, our approach is easy to adapt to other Xilinx series such as XC2000 and XC3000 with minor modifications.

4 Chip Estimation

4.1 Problem Statement

Given an RT level description, the goal of Chip Area Estimation is to predict the area of the chip in terms of number of CLBs as well as the most appropriate device it may fit by using the area information of all the RT level components.

Given an RT level description, the goal of Chip Timing Estimation is to estimate the performance of the chip in terms of minimum clock period by using the delay information of all the RT level components along with the estimated topology information obtained from Chip Area Estimation.

4.2 Chip Area Estimation

Our chip level area model uses a slicing tree techniques derived from [8] for evaluating the area of designs implemented using RT level components.

4.2.1 Component Shape Function

To improve the density of the chip, designers may try different floorplans by varying the topological placements of each component. Component *shape function* represents the different topological placements in the actual layout and their corresponding delay information [10].

At the RT level, the shape functions of some components can be obtained from our component library which is a collection of hard macros with shape function and delay information. The collection of hard macros includes components those are frequently used in the design so we pre-characterized their shape function. Also, it includes vendor supplied pre-designed components such as hard macros in Xilinx library etc. For the components whose shape function is not known *a priori*, (controller for example), their shape function can be obtained by invoking the Component Estimator, CompEst-FPGA, described in [9]. CompEst-FPGA estimates the area and delay of combinational circuits described either at the gate level or using boolean equations. It estimates the outcome of the technology mapping, placement, routing, and timing optimization phases of the design procedure. CompEst has been benchmarked

with respect to a wide variety of gate level designs with and without post layout optimization. The results indicate that the estimation is not only accurate (10-15% error in timing estimation and 5% in area estimation), but also time efficient, taking 1-2 orders of magnitude less runtime to evaluate compared to the actual Xilinx design tools.

4.2.2 Chip Level Area Model

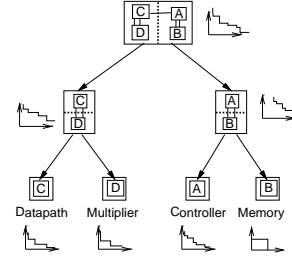


Figure 4: Constructive/analytical area estimation technique

The chip level slicing tree technique involves slicing down to the leaf blocks which consists of either RT level components or controller. This constructive approach does not consume excessive runtime since the number of leaf blocks are limited to a relatively small number. This technique is illustrated in Figure 4. The slicing tree is built by recursively partitioning the input design. Because of specific characteristics of FPGA, partitioning objectives have to be selected accordingly. One of them is minimization of routing resource consumption. It is mainly accomplished by devising data objects that will partition in such a way as to permit the greatest number of signals to traverse the shortest distances along the fewest routing channels with the least crossovers. This most often means placing interconnected objects adjacent to each other with related elements aligned to the routing axes.

Because of the granularity of FPGA (the area is in terms of CLBs rather than in terms of micron, e.g. the Xilinx XC4013 has 24 by 24 CLBs rather than thousands by thousands square microns in the custom design), reducing unused area is a very important objective. To achieve this, objects with similar sizes are placed adjacent to each other because this can minimize the wasted area. Sometimes, this will conflict with the objective to put strongly connected blocks adjacent to each other. We introduce a *cutting edge threshold* in our algorithm to trade off between area and performance. The cutting edge threshold actually is a parameter obtained by calculating the average size of all the blocks to be partitioned, if some block's size exceeds the cutting edge threshold (that means it is far bigger than the rest of the blocks, it will be isolated from the rest of the blocks and be a sub-slice of the current slice. For example shown in Figure 5(a), the netlist contains 4 components, Mult needs 60 CLBs, two registers need 16 CLBs each, one Mux needs 8 CLB. If we only consider the interconnection between them, we will end up with a 12x12 CLB device as shown in Figure 5(b), if we consider the cutting edge threshold, the Mult will be isolated from the rest of the blocks and be one sub-slice for slice 1234. The result with the cutting edge threshold is a 10x10 CLB device as shown in Figure 5(c), we can see that slicing with the cutting edge threshold produces more area-efficient result.

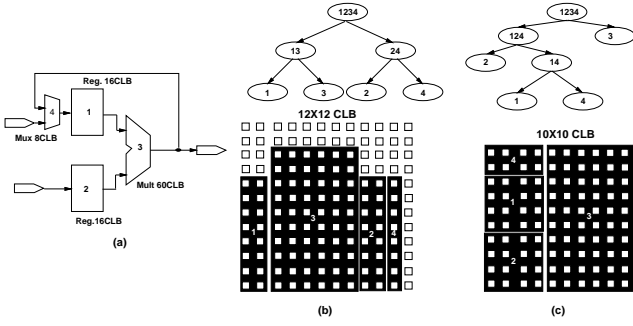


Figure 5: An Example:(a) Netlist; (b) Slicing without the cutting edge threshold;(c) Slicing with the cutting edge threshold

The shape function of the entire design is computed by constructively adding the shape function of these leaf blocks. In addition to the area of leaf blocks, the routing area used by the nets connecting these blocks also needs to be accounted for. 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 depend on the shapes and sizes of the two sibling blocks. 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 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. When the required routing resources exceed the budget, the size of the composite block (formed by combining the two sibling blocks) will correspondingly be increased so as to accommodate the extra routing requirements. Additionally, all the parent blocks in the slicing tree are correspondingly adjusted as well.

At the end of this phase, we can estimate the area of the overall chip, according to the number of I/O, we can predict whether the design can be fitted into one FPGA device or not, if it can be fitted, we can also predict the specific XC4000 device which will be the best choice. Let W, H, num_io be the estimated width, height, and number of IOs of the chip respectively, $W_1, W_2, H_1, H_2, num_io_1, num_io_2$ be the width, height and number of IOs of two consecutive devices: $device_1, device_2$ respectively.

if

$$((W_1 < W \leq W_2) \text{AND} (H_1 < H \leq H_2) \text{AND} (num_io_1 < num_io \leq num_io_2)) \quad (1)$$

then $device_2$ is the best choice.

At this moment, we also have an approximate topology of the chip which can be used in the subsequent timing models described next Section ⁵.

⁵For more details, the reader is referred to [10]

4.3 Chip Level Timing Estimation

The Chip delay includes component delays, wire segment delay, and Programmable Interconnection Point (PIP) delay. The Chip timing estimation model includes predict the pin location on each leaf block, predict wiring delay and predict chip clock cycle three phases.

4.3.1 Predict the Pin Location on Each Leaf Block

Given an input RT level design, our chip level area model described in Section 4.2.2 outputs an approximate floorplan which provides estimates of the relative locations of the constituent blocks. To better estimate chip level timing, pin location must be either known or estimated. On those blocks which have been pre-designed, the pin location are known. For other components which have not been laid-out yet, we must estimate “preferred” location for each pin. location can be determined by evaluating the approximate topology of the design. Chip area estimation process determines the approximate locations of the blocks in the design taking routing area into account. For each net, first, we identify the source pin, then we identify load pins and their associated blocks. By evaluating the mean location of these blocks, a “preferred” side location of each source pin is first determined. Then, by finding the shortest Manhattan distance between each pair of source and destination blocks, a preferred location of each sink pin can also be determined.

4.3.2 Predict wiring delay

To predict the delay between point A and B, $D(A, B)$, in Figure 6, the Manhattan distance x and y values (in units of CLBs) are first calculated. Then, a wire type (single-length line, double-length line and long line) is assigned to that wire as described in the following section. This decides the number of PIPs and number of segments between points A and B. Subsequently, the point-to-point delay (pin-to-pin 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 $D_{pp}(A, B)$ with a fanout factor as described below.

point-to-point delay:

for single/double length lines ---
 $D_{pp}(A, B) = \#PIP \times d_{pip} + \#seg \times d_{seg}$

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

d_{seg}	d_{pip}	$d_{ }$
0.3	0.4	0.18

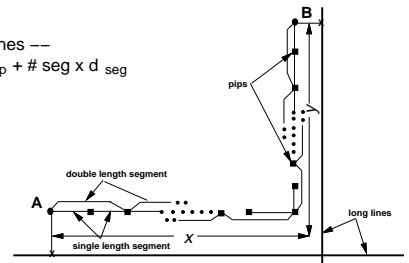


Figure 6: Point-to-point delay model and associated parameters.

To predict the wire type, the algorithm mainly checks the interconnect wire length x and y respectively. First, long lines are assigned to all the wires which are longer than 8 CLBs in either direction. Then, single-length lines are assigned for all wires which are shorter than 2 CLBs. Note that single-length lines can not be connected to double-length lines. Thus, if one segment of a wire is assigned to

a single length line, then the other segment of the wire is also assigned to a single length line if its length is between 2 and 8 CLBs. Finally, double-length lines are assigned to the rest of the interconnect wires.

From Section 2.2, 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 same⁶. 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 Manhattan distances of (A, B) in X and Y directions respectively (both in units of CLBs). If only single-length lines are used, they will pass through $x+1$ and $y+1$ PIPs, and through x and y 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 (pin-to-pin delay) will be:

$$D_{pp}(A, B) = \begin{cases} d_{seg} * x + d_{pip} * (x + 1) + \\ d_{seg} * y + d_{pip} * (y + 1) & \text{for SLs} \\ d_{seg} * \lfloor \frac{x}{2} \rfloor + d_{pip} * \lfloor (\frac{x}{2} + 1) \rfloor + \\ d_{seg} * \lfloor \frac{y}{2} \rfloor + d_{pip} * \lfloor (\frac{y}{2} + 1) \rfloor & \text{for DLs} \\ d_{ll} * (x + y) & \text{for LLs} \end{cases}$$

and the associated parameters are listed in Figure 6.

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 2.5. we can see that the fanout delay effect at the chip level is quite big. This is because at the chip level, part of fanout effect could be masked by the components. For example in Figure 7, net n fans out from block A to two other

blocks, B and C, so its RT level fanout is 2. However, the net actually feeds 5 CLBs when the design is flattened.

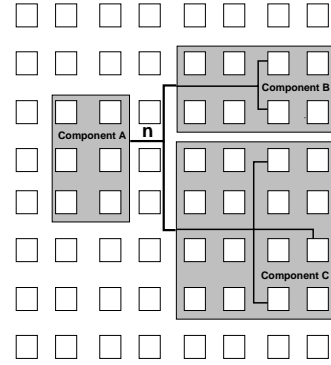


Figure 7: Fanout Effects

At the end of this step, we have a netlist which contains components’ delay and the estimates of net delay.

4.3.3 Predict Clock Cycle Length

A typical timing model for digital systems is shown in Figure 8. The datapath part is composed of datapath logic blocks and the data registers. Data registers are used to store data inputs, outputs, and intermediate values in the data path. Our timing model assumes that the controller is implemented as a Moore Finite State Machine. A Moore controller consists of two combinational logic blocks: the next state logic and the output logic, one or more control registers store the current state information. The data path consists of combinational logic blocks (composed of functional units and muxes) bounded by data registers⁷.

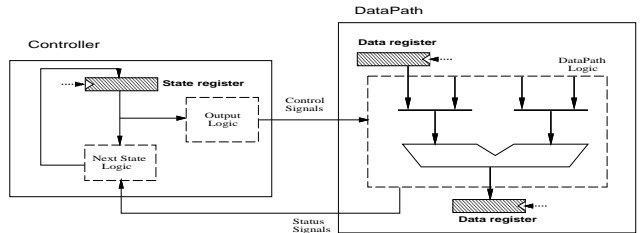


Figure 8: Typical Timing Model for a Digital System

Thus, the overall system can be modeled as a network of combinational logic blocks separated by registers. In this case, the worst case register-to-register delay is estimated and is output as a lower bound on the clock period for single phase clocking.

The total execution time of a design is given as the number of time steps times the clock period. The number of time steps is determined by scheduling and allocation and is known once the RT level design is generated. The minimum possible clock period is determined by the worst case register-to-register delay. Note that our timing models are kept simple due to runtime efficiency constraints. Our goal here is not to provide accurate timing analysis of the design. Rather, the aim is to provide the higher level tools

⁶The model can be easily modified to account for different delays of single-length and double-length segments, if needed.

⁷This assumption, however, does not affect the validity of the overall approach since it is possible to substitute different timing models for other types of controllers should that be necessary.

with an early assessment of design cost and performance. However, the designer can easily apply more accurate timing analysis models using the delay estimates of the various blocks and interconnections which are produced by ChipEst-FPGA (i.e. a *forward annotated* RT level netlist).

5 Experimental Results

In order to benchmark the accuracy of our ChipEst-FPGA, we used six benchmark designs: (1) the AMD 2901 cpu with a bitwidth of 4, (2) RISC microprocessor Zot1 [7] with 15 instructions and data path bitwidth is also 4, (3) The Differential Equation Example (HAL), (4) the Elliptic Filter [11] which with a bitwidth of 4 and 13 time steps. (5) and (6) are Fuzzy logic examples derived from [1]. Altogether, the RT-level implementations spanned a reasonably large set of design variation that are likely to be considered during high level design. The FPGA chips vary from XC4005 (with 12x12 CLBs) to XC4010 (with 20x20 CLBs).

All the RT-level implementations were written in VHDL. For components that can be pre-characterized, we can obtain their layout and timing information from the library. The layout and timing information for the remaining of components either (1) by invoking our area and timing estimation described in [9] or (2) by actually implementing the components. Clearly, the accuracy of the chip level estimation will vary if procedure (1) is followed, but the overall estimation procedure will be more runtime efficient since uncharacterized components such as controller can be estimated “on-the-fly” by CompEst-FPGA. Since we are interested in benchmarking the chip level estimation procedure at this point, we use procedure (2) by designing each component as shown in [9] rather than run CompEst-FPGA (Procedure (1)) to get the actual layout and timing information.

The Chip level design is finished by instantiating components as hard macros with specific layout and timing information. Once we got the chip xnf files, again, they are fed into xilinx ppr and Xdelay are used to get the delay for the whole chip. Because of the non-deterministic nature of ppr, the designer tends to run ppr many times with different seeds and select the best one (in the experiments we ran, the worst delay varied from 4.4% to 20.9% percent off the best case in ten runs). To be fair, we also pick the layout with best performance to compare with our estimated results. In our experiments, ppr and Xdelay are run 10-20 times with different seeds and the best design is selected for comparison.

In order to assess the accuracy of our chip level estimation, we feed same RT level VHDL file into our ChipEst-FPGA to produce estimates of the chip area and delay using the models described in previous sections.

The estimation results are shown in Figure 9. First, we note that our area estimates are very accurate. Our estimation accurately predicted the exact device type needed every time. For performance estimation, there was some differences between estimated and measured values. These differences can be attributed to the following factors: (1) differences between estimated and final placements; (2) differences between routing rule assignment and final routing; (3) inaccuracies in the wiring delay model. Our ChipEst-FPGA can produce highly accurate estimates within very short runtime. The average estimation error for perfor-

mance is about 5.1%, while the worst case error is 18.7%. Even when one run of ppr/Xdelay is assumed, our estimation is still at least an order of magnitude faster to obtain than the actual layout process. This clearly indicates that our tool can be efficiently used to provide fast and accurate feedback to synthesis tools, allowing them to make better informed design decisions.

Benchmark design	Measured device (area CLB)	Estimated device	Measured I/Os	Estimated I/Os	Measured clock cycle (ns)	Estimated clock cycle (ns)	% error (cycle time)	Estimation run time(1) (s)	PPR/ Xdelay(2) (s)
AMD 2901	XC4006 (16x16)	XC4006 (16x16)	40	40	181.7	181.1	+0.4	1.8	8340
Zot1	XC4005 (14x14)	XC4005 (14x14)	51	51	99.5	118.1	+18.7	6	319
HAL	XC4006 (16x16)	XC4006 (16x16)	32	32	173.9	179.1	+3.0	3.8	327
EF19	XC4010 (20x20)	XC4010 (20x20)	66	66	489.6	482.2	-1.5	10.8	1835
Fuzzy1	XC4008 (18x18)	XC4008 (18x18)	76	76	286.6	272.1	-4.8	5.5	413
Fuzzy2	XC4006 (16x16)	XC4006 (16x16)	77	77	287.5	281.8	-2.0	5.2	566
Average % error								5.1	
(1) CPU run time. (2) reported by ppr and xdelay.									

Figure 9: Experimental Results

6 Conclusion

We presented a set of area and delay estimation techniques to support a hierarchical design model for Lookup Table Based FPGAs. The overall approach was benchmarked and found to be accurate. Future work will concentrate on linking the estimation model to synthesis so that better quality designs can be produced.

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.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.
- [3] Xilinx, “XACT Development System: Libraries Guide,” *Xilinx*, 1994.
- [4] 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.
- [5] Xilinx, “XACT Xilinx Synopsys Interface FPGA User Guide,” *Xilinx*, 1995.
- [6] Robert J. Francis, “A Tutorial on Logic Synthesis for Lookup-Table Based FPGAs,” *Proc. ICCAD 92*, 1992.
- [7] D. Craig, M. Pontius, “The Zot1 Microprocessor implemented on an FPGA,” *UCI course project report*, 1994.
- [8] X. Chen and M. L. Bushnell, “A module area estimator for vlsi layouts,” *Proc. 25th Design Automation Conf.*, pp. 54-59, IEEE/ACM, 1988.
- [9] M. Xu, F.J. Kurdahi, “Area and Timing Estimation for Lookup Table Based FPGAs,” *Proceeding of European Design & Test Conference*, 1996
- [10] M. Xu, F.J. Kurdahi, “Chip Level Area and Timing Estimation for Lookup Table Based FPGAs,” *Technical Report #95-31, UCI*, Aug.1995.
- [11] S.Y. Kung, H. J. Whitehouse, and T. Kailath, “VLSI and Modern Signal Processing.” *Prentice Hall*, 1985.
- [12] D.D. Gajski and F. Vahid. “A system design methodology: Executable-specification refinement.” *Proc. of the European Conference on Design Automation (EDAC)*, 1994.