Towards Scalable FPGA CAD Through Architecture

Scott Y.L. Chin and Steven J.E. Wilton
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, British Columbia, Canada
scottc@ece.ubc.ca, stevew@ece.ubc.ca

ABSTRACT

Long FPGA CAD runtime has emerged as a limitation to the future scaling of FPGA densities. Already, compile times on the order of a day are common, and the situation will only get worse as FPGAs get larger. Without a concerted effort to reduce compile times, further scaling of FPGAs will eventually become impractical.

Previous works have presented fast CAD tools that trade-off quality of result for compile time. In this paper, we take a different but complementary approach. We show that the architecture of the FPGA itself can be designed to be amenable to fast-compile. If not done carefully, this can lead to lower-quality mapping results, so a careful tradeoff between area, delay, power, and compile run-time is essential. We investigate the extent to which run-time can be reduced by employing high-capacity logic blocks. We extend previous studies on logic block architectures by quantifying the area, delay and CAD runtime trade-offs for large capacity blocks, and also investigate some multi-level logic block architectures. In addition, we present an analytically derived equation to guide the design of logic block I/O requirements.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate Arrays*

General Terms

Design

Keywords

Field-Programmable Gate Arrays, FPGAs, Architecture, CAD Run-time

1. INTRODUCTION

Since their introduction, Field-Programmable Gate Arrays (FPGAs) have grown substantially in capacity due to the continued scaling of transistor sizes. This has opened up new markets for FPGAs since they can now implement more complex digital systems. However, this scaling also means that the FPGA CAD tools need to perform an increasing number of operations to complete the mapping process. For modern FPGAs, place and route for even a moderate-sized design can take an entire workday. As FPGA capacity continues to scale, the problem will become worse. To exacerbate the problem, the computer processors used to run the CAD tools are no longer scaling in speed with transistor size and parallel CAD implementations provide sub-linear speedups. This is not sustainable. Left unaddressed, further scaling of FPGA capacity will eventually become impractical due to CAD runtime.

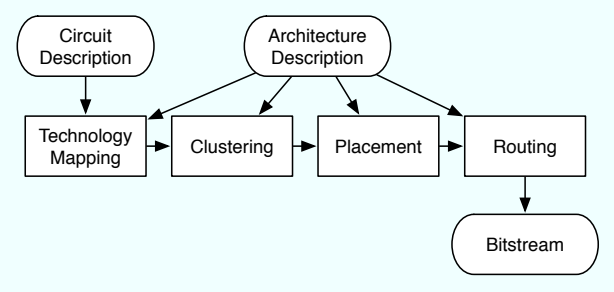

Figure 1: Typical FPGA CAD Flow

To combat this problem, significant advances have been made to improve the efficiency of FPGA CAD algorithms. However, tackling this issue should not be left solely to CAD developers. Since architecture and CAD are tightly connected, it is conceivable that architecture designers can design architectures that are amenable to fast CAD runtimes, balanced against the more traditional optimization goals of power, delay, and density. This approach is a large unexplored area of research and the techniques from this approach may be orthogonal, and hence complementary, with existing and future algorithmic research.

We can identify two general approaches for designing such architectures:

1. *Architectures with features to recoup degraded mapping quality of fast CAD algorithms.* Most fast CAD techniques lead to degraded mapping quality (i.e. density, delay, and or power) [18]. It is conceivable to over-design or optimize the architecture to compensate for this degradation. The end goal would be to achieve equivalent mapping quality and faster CAD at the ex-

FPGA'11, February 27–March 1, 2011, Monterey, California, USA.

pense of absolute area. The Plasma architecture is one example of this approach [3].

2. *Architectures that lead to a reduced problem size in the computationally expensive stages of the FPGA CAD flow.* The FPGA CAD flow contains a number of stages. The more prominent stages are shown in Figure 1. Some stages, such as placement and routing, take much longer than others. It may be possible to modify the architecture so that the problem size presented to the computationally expensive stage(s) is reduced, thus directly reducing the runtime of these stages. This may come at the expense of increasing the problem size for other (less computationally intense) stages, or require the introduction of whole new stages to the CAD flow.

In this paper, we focus on the second approach, and *investigate to what extent it is possible to architect high-capacity logic blocks to reduce CAD runtime.* More specifically, we consider two approaches for building high-capacity logic blocks: directly scaling traditional architectures, and using a multi-level architecture. We explore logic block sizes beyond the range looked at in previous studies and approach this study from the new perspective of CAD runtime. In addition, we present a new analytical relationship for determining the number of inputs per logic block which accounts for interconnect demand.

The paper is organized as follows: Section 2 discusses background the related work. Section 3 describes the logic block architectures explored in this paper. Section 4 presents the experimental methodology Section 5 discusses the results of our experiments. And Section 6 concludes the paper.

2. BACKGROUND

2.1 Related Studies in CAD Runtime

Reducing the time needed to map a circuit to the FPGA architecture has garnered much research interest. Most of these studies aim to improve the efficiency of the underlying CAD algorithms [4, 23, 5]. More recently, there has been interest in parallelizing the CAD algorithms [14] to take advantage of the increasing number of processor cores on a single CPU.

Some methods impact the FPGA user's design flow. Incremental compilation [21] allows the FPGA user to re-map only the portion of the circuit that has been changed as opposed to remapping the entire circuit. Floorplanning leverages the fact that large-scale designs are often developed using pre-defined macroblocks or using hierarchical hardware design languages [22]. This may allow the mapping of each macroblock to occur independently, or provide additional locality information to help the mapping process converge faster.

These previous techniques focus on *algorithmic* modifications; in this paper, we investigate *architectural* modifications that may be orthogonal to these CAD approaches.

2.2 Logic Block Architecture Terminology

Logic blocks (LB) in modern FPGAs are implemented as clusters of lookup tables (LUTs) and registers, and include additional specialized circuitry to perform clock distribution, control (register set and reset), and fast arithmetic operations. For simplicity, we assume the commonly used view

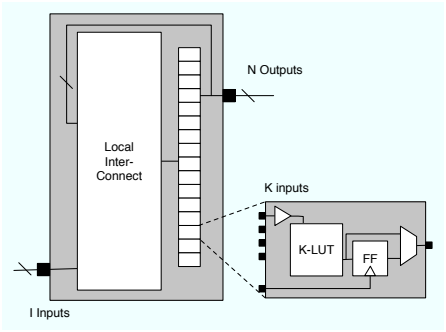

Figure 2: Logic Block and Basic Logic Element (BLE)

shown in Figure 2. Each LB consists of a fast local interconnect network and a group of basic logic elements (BLEs). Each BLE contains a LUT, a register, and a programmable multiplexer (MUX), and each BLE output can drive signals outside of the LB. This architectural model can be used to describe a family of architectures using the following three parameters:

- K denotes the size of each LUT
- N denotes the number of BLEs within a logic block
- I denotes the number of unique inputs that can be supplied to the logic block

2.3 Previous Studies on Logic Block Size

Architecture studies have been performed to investigate the impact of LB size on area, delay, and power. When optimizing the architecture for area [4, 17, 1] or power efficiency [20], an LB size in the range of $N = 4$ to 9 is shown to be best. Three competing trends lead to this optimal range. As LB size increases, the area and power dissipation of each LB increases. However, the total number of LBs needed to implement a circuit decrease. The amount of global routing resources is also reduced as more nets are completely absorbed and implemented by the local interconnect.

The critical path delay of a circuit is also affected in a similar way. Delay through a single LB increases, but the total number of LBs along the critical path decreases and more nets are implemented using the faster local interconnect. Unlike area and power, the studies showed that the latter trend dominates. Thus, critical path delay decreases monotonically for increasing cluster size.

Due to the area and power results, there has never been a compelling reason to investigate large LB sizes (commercial devices use a LB size in the range of $N = 8$ to 16). Now that CAD runtime is becoming a major concern (its relative importance to area, delay, and power is still debatable) we have a reason to revisit these topics.

Although [17] observed that logic block size affects CAD runtime, none of these studies directly investigate the CAD runtime versus quality (area, delay, power) trade-offs afforded by logic block size. In addition, none of these studies investigate an LB size greater than $N = 20$ [1].

[1] [1, 20] looked at sizes up to $N = 10$ for a 180um process, [4] and [17] looked at sizes up to $N = 16$ and $N = 20$, respectively, for a 350um process.

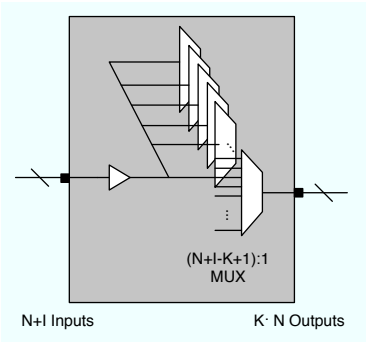

Figure 3: Logic block local interconnect

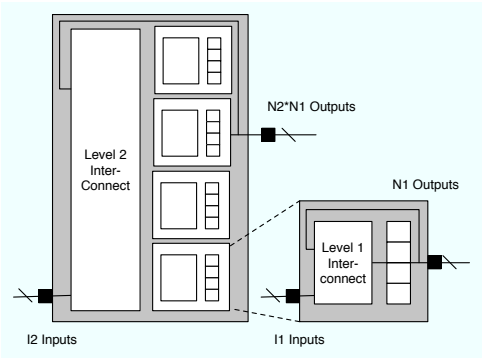

Figure 4: Multi-Level Logic Block

2.4 Why Logic Block Size Affects Runtime

Intuitively, as the logic block size increases, the logic capacity of each LB is increased. This has two major effects on CAD runtime:

1. The number of logic blocks needed to implement the user circuit is reduced, thus directly reducing the placement problem size and runtime.

2. The number of nets that need to be routed outside of the logic blocks is reduced, thus directly reducing the routing problem size and runtime.

These effects have been observed experimentally [17] and expressed analytically [6].

Of course, increasing logic block size will increase the runtime of the CAD stages that map logic to these blocks. However, the run-time of these stages is much smaller than the run-time of the placement and routing stages.

3. HIGH-CAPACITY LOGIC BLOCK ARCHITECTURES

In this paper, we consider two methods to architect large logic blocks. The first approach is to directly increase N in the logic block architecture that we have discussed so far. We will refer to this approach as *flat*. The second approach will use a partly hierarchical construction with two levels of hierarchy. We will refer to this as *multi-level*. This is similar to the APEX logic block architecture [2]. We do not consider fully hierarchical architectures or any further levels of hierarchy within this paper.

3.1 Flat Logic Block

Section 2.2 has already described this architecture. In our study, we consider sizes up to $N = 50$. We assume a depopulated LB local interconnect using the technique from [24] that maintains full logical connectivity, but reduces the number of transistors needed to implement the network. This scheme guarantees that at least one of the K BLE inputs can be connected to any of the I logic block inputs, and is possible due to the logical equivalence of the LUT input pins. The effect is a reduction in width of the $N \cdot K$ local routing MUX's (one for each BLE input pin in the logic block) from $(N + I) : 1$ to $(I + N - K + 1) : 1$. The local interconnect is illustrated in Figure 3.

3.2 Multi-Level Logic Block

This architecture consists of two levels. The first level is the same as the flat architecture from the previous section. We use K, $N1$, and $I1$ to denote the architecture parameters of the first level. The second level of hierarchy consists of a second level of interconnect and $N2$ first-level blocks. The number of inputs available at the second level is parameterized as $I2$. Figure 4 shows this architecture.

We chose this architecture because it fits easily into an island-style FPGAs and does not require any major changes to the global routing architecture or CAD tools. Investigating more radical alternative architectures is an interesting area of future research.

3.3 Logic Block I/O and Interconnect Demand

Determining the appropriate number of inputs (I, $I1$, and $I2$) for the LB is critical to the area and packing efficiency of the architecture. Too few inputs force BLEs to go unused and too many lead to an unnecessarily large silicon area due to the size of the local interconnect.

The total number of BLE inputs within a LB is $K \cdot N$. But, due to shared and feedback connections (which are implemented by connections already inside the LB) in the circuits being implemented, LBs typically do not need the $K \cdot N$ maximum number of unique inputs.

Betz [4] found that the relationship shown in Equation 1 allowed 98% of BLEs to be used on average. This relationship was determined empirically when investigating architectures of $K = 4$ and N in the range of 1 to 16. It was later generalized by Ahmed [1] based on experimental data using architectures of $K = 2$ to 7 and $N = 1$ to 10. This equation has since become a well-accepted rule of thumb.

$$I = \frac{K}{2}(N + 1) \tag{1}$$

We found that this relationship becomes less appropriate when scaling to very large LB sizes. The main shortcoming is that it does not directly account for the the interconnect demand of circuits being implemented on the FPGA. The end result is a higher than necessary number of input pins per LB. Using analytical methods from [12], we derive a more accurate relationship that accounts for interconnect demand of the circuits being implemented on the FPGA.

Starting with the same goal of the traditional equation, we aim to determine a value I such that 98% of the BLEs are utilized on average. Let us denote the number of K-LUTs

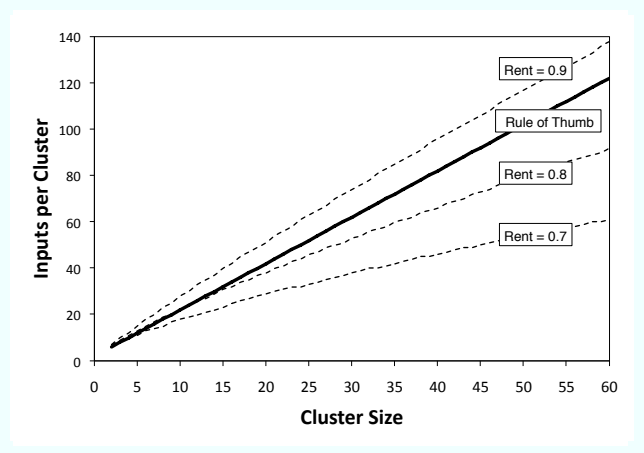

Figure 5: Equations for determining number of inputs per logic block

required to cover a circuit as n_k. We then denote the number of LBs needed to cover these K-LUTs as n_c. Each LB contains N K-LUTs so the total number of programmable K-LUTs available across all LBs is $n_c \cdot N$. If only 98% of the available K-LUTs are used, we can then form the following equality:

$$n_k = 0.98 \cdot n_c \cdot N \tag{2}$$

We then use Equation 3 directly from [12] [2]. This equation models the number of clusters (logic blocks) required, denoted as n_c, to implement a circuit that has been technology mapped to n_k LUTs. Note that this equation accounts for the interconnect demand of the circuit being implemented; p and f denote the Rent parameter and average fanout of the circuit, respectively. γ is a model parameter that describes the average number of unused LUT input pins. The full derivation can be found in [12], and its accuracy has been validated against experimental results.

$$n_c = n_k \cdot \sqrt[p]{\frac{K + 1 - \gamma}{I \cdot \left(1 + \frac{1}{f}\right)}} \tag{3}$$

Substituting Equation 3 into Equation 2 and solving for I yields Equation 4.

$$I = (0.98 \cdot N)^p \cdot \frac{K + 1 - \gamma}{1 + \frac{1}{f}} \tag{4}$$

We can interpret the Rent parameter p in Equation 4 as the Rent parameter of the architecture. In other words, it describes the architecture's *interconnect supply*. Figure 5 compares the traditional rule-of-thumb equation (shown in the solid line) to the new analytical equation (shown in dashed line for three values of interconnect supply. Note that we used a fanout value of of $f = 3.4$ based on the average fanout of our benchmark suite presented later in Section 4). From this figure, we can see that beyond $N = 15$, the traditional equation produces architectures with an intrinsically high amount of interconnect supply.

[2]The model in [12] contains two equations for n_c depending on whether the logic architecture leads to N-Limited, or I-Limited packing. In our scenario, we state that only 98% of BLEs are used because I is chosen in a way to limit full occupancy. Therefore, we use the I-Limited equation

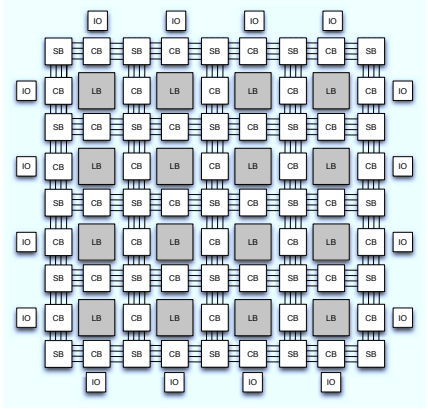

Figure 6: FPGA Overview

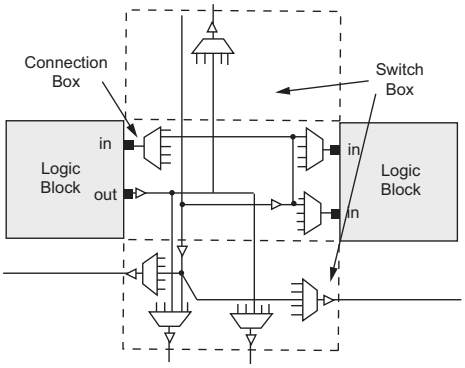

Figure 7: FPGA Routing Architecture Overview

As a reference point, previous studies have quantified the architecture Rent exponent of the Altera Cyclone architecture and Altera Apex architecture to be 0.7256 [19], and 0.78 [11], respectively. Those studies also quantified the interconnect demand of 77 industrial circuits and found the Rent parameter to range from 0.5-0.8 with an average of 0.6.

For the remainder of this paper, we will use this relationship to determine the number of input pins per logic block. For the multi-level architecture, we determine $I1$ by substituting $N1$ into the equation, and we determine $I2$ by substituting in the *effective* cluster size $N2 \cdot N1$.

4. EXPERIMENTAL METHODOLOGY

In the next section, we will experimentally quantify the area, delay, and CAD run-time trade-offs for the architectures described in Section 3. This section will describe the experimental methodology including architectural assumptions, area and delay models, CAD tools and benchmarks.

4.1 Architectural Assumptions

We assume a homogeneous FPGA consisting only of logic block tiles with a single clock domain. The routing archi-

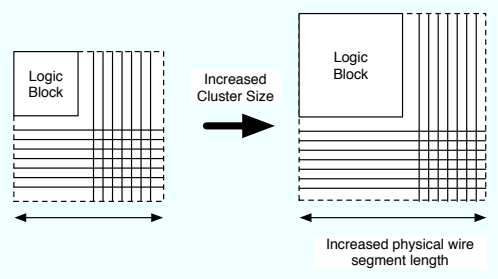

Figure 8: Effects of Cluster Size on Physical Wire Segment Length

tecture is based on single-driver directional routing which is consistent with VPR 5.0. Figure 6 and 7 show an overview.

Each LB output pin drives an isolation buffer which then drives $Fcout \cdot W$ routing MUXes, where W is the number of routing tracks per channel and $Fcout$ is the percentage of W tracks to which the the output pin can connect. Wire segments connect to LB inputs through an isolation buffer which then drives $Fcin \cdot (2 \cdot I/4)$ input pins, where $Fcin$ is the fraction of wire segments in the routing channel that can drive a given input pin. We assume that LB pins are evenly distributed along all four sides of the LB. Therefore, there are $I/4$ input pins on each side, and two LBs surrounding each channel. Wire segments entering a Switch Box can connect to Fs outgoing wire segments.

In our experiments, we set $Fcout = 0.1$, $Fcin = 0.15$, $Fs = 3$ and assume that wire segments span a single LB. The channel width W is set to the minimum width that still enables the circuit to be routed. This is determined independently for each circuit and each set of logic block parameter values. Similarly, the FPGA grid size is set just large enough to accommodate all of the required LBs for each circuit. Inputs per LB is chosen based on a Rent parameter of $p = 0.8$. We performed our experiments for $p = 0.7$ and $p = 0.9$ as well but have omitted the results since the trends and conclusions were similar.

I/O blocks are found on the periphery of the FPGA. Each I/O block contains a number of pads that allow for off-chip communication. To maintain a similar I/O Pad to I/O Block density across all our architectures, we set the number of pads per block according to Equation 5 [17].

$$PadsPerIOBlock = \lceil 2 \cdot \sqrt{N_{eff}} \rceil \quad (5)$$

For the flat architecture, $N_{eff} = N$ and for the multi-level architecture, $N_{eff} = N2 \cdot N1$.

4.2 Area Model, Delay Model, and Circuit Design

We use an area model based on counting the number of minimum-width transistor areas (MWTA) needed to implement the architecture [4]. In the model, SRAM cells are implemented with six minimum-size transistors. LUTs are implemented as full binary trees of NMOS pass-transistors and all pass transistors are minimum width. Multiplexers (MUX) are implemented as two-stage pass transistor trees, which is consistent with VPR's assumptions. Routing SBOX drivers are implemented as three stage buffers similar to [13]. All other buffers are two-stage. Buffers are sized to minimize

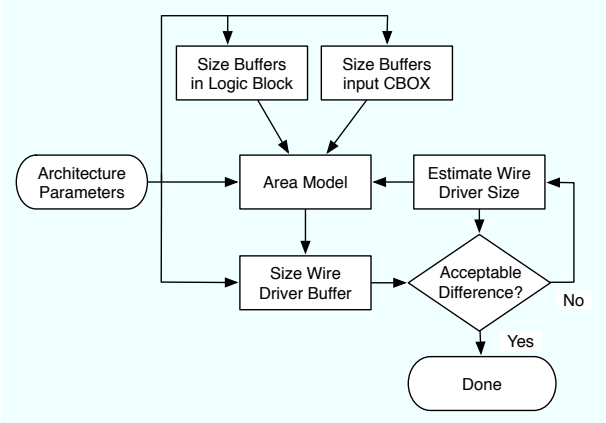

Figure 9: Buffer Sizing Flow

delay using HSPICE automatic-sizing. We use circuit-level designs similar to those described in [10]. We assume a 45nm process througout and use the Predictive Technology Model [25] to acquire process information for HSPICE.

As the size of a LB increases, the physical distance between clusters also increases. This directly affects the physical length (as shown in Figure 8) of the routing wire segments which in turn affects the SBOX driver sizes. To account for this effect, we use the iterative flow show in Figure 9. We first size all buffers except for the wire driver. These sizings, along with an estimate of the SBOX driver size, are then supplied into the area model to obtain a MTWA count for a single tile in the FPGA. We then convert this count to a physical area (and wire length) assuming a 60% layout density [4] and size the SBOX drivers. This repeats until the estimate for the SBOX driver size becomes acceptable.

4.3 CAD and Benchmarks

We use a CAD flow similar to [17, 1]. First, benchmark circuits are technology mapped to LUTs using Flowmap [7]. We then use a timing-driven packing algorithm T-VPack [17] to pack LUTs and registers into logic blocks. Placement and routing is performed using VPR 5.0 [15].

In our experiments with multi-level cluster architectures, we apply two iterations of the T-VPack algorithm to pack a circuit. Although specialized multi-level packing algorithms have been proposed [8, 9], we focus on the trade-offs in the architecture by keeping the same algorithm across both the flat and multi-level packing. To perform timing analysis during place and route for the multi-level LBs, we have made straight-forward extensions to the timing-graph construction in VPR's code. The core timing analysis algorithms are unchanged.

Runtime measurements are performed on Xeon 2.6GHz processors. When quantifying the routing runtime, we only measure the time needed to route to the minimum channel-width architecture (as opposed to measuring the entire binary-search routing process needed to determine the minimum channel width).

We use the synthetic benchmarks listed in Table 1 for most of the experiments. We chose to use synthetic circuits because the circuits in the traditional MCNC benchmark suite are too small for this study. The synthetic benchmarks mimic modern system-level designs and were generated us-

Table 1: Synthetic Benchmark Circuits

Circuit	4-LUTs	Nets	Rent	Avg. Fanout
synth01	15818	38047	0.654	3.3023
synth02	8831	21288	0.658	3.3745
synth03	10906	26598	0.641	3.5053
synth04	20007	48460	0.631	3.4597
synth05	18115	42917	0.609	3.3955
synth06	11786	28300	0.657	3.4675
synth07	26461	63934	0.648	3.4377
synth08	20961	50008	0.646	3.4388
synth09	12085	29020	0.659	3.3614
synth10	10009	24014	0.648	3.3776
synth11	16867	38661	0.605	3.4438
synth12	13131	31161	0.692	3.3252
synth13	3160	7973	0.648	3.3799
synth14	12405	31085	0.689	3.1425

ing the tool published in [16]. We use the entire MCNC suite of circuits to form the IP library needed by the circuit generation tool. All generated circuits are sequential.

We repeat some of the experiments using the smaller MCNC20 suite shown in Table 2 to highlight the effects of circuit size on some experimental conclusions.

Table 2: MCNC Benchmark Circuits

Circuit	4-LUTs	Nets	Rent	Fanout
alu4	1522	1536	0.630	3.362
apex2	1878	1917	0.640	3.442
apex4	1262	1271	0.683	3.307
bigkey	1707	2194	0.626	3.910
clma	8381	8797	0.537	3.336
des	1591	1847	0.699	2.939
diffeq	1494	1935	0.574	3.064
dsip	1370	1823	0.673	2.820
elliptic	3602	4855	0.643	2.944
ex1010	4598	4608	0.618	3.344
ex5p	1064	1072	0.700	3.480
frisc	3539	4445	0.638	3.395
misex3	1397	1411	0.642	3.437
pdc	4575	4591	0.674	3.759
s298	1930	1942	0.542	3.851
s38417	6096	7588	0.603	2.914
s38584	6281	7580	0.612	2.889
seq	1750	1791	0.662	3.417
spla	3690	3706	0.639	3.704
tseng	1046	1483	0.603	3.123

5. EXPERIMENTAL RESULTS

5.1 Directly Scaling N

Figure 10 shows the effects of LB size on area, delay, and CAD runtime using the synthetic benchmark suite. Each data point is the averaged over the entire suite. For area and delay, both logic and routing components are shown along with the total. For CAD runtime, both placement and routing runtime are shown along with the total runtime.

5.1.1 Area

The impact on area agrees with previous studies. Logic area increases as the area of each LB grows. The routing area decreases until approximately $N = 25$ as more and more nets are completely absorbed into the LB. Beyond this point, the routing area increases for two reasons. First the number of pins on an LB increases which means that an increasing number of nets need to connect to each LB. This makes it challenging for the placement algorithm to minimize wirelength since moving a single block perturbs many nets. The routing also becomes more challenging as congestion density increases, leading to the need for more routing tracks per channel.

One difference from previous studies is the optimal range for LB size. In this study, $N = 14$ to 18 leads to minimum area. We repeated the experiment using the smaller MCNC20 benchmark suite and the area results are shown in Figure 11a. The results with the smaller circuits gives an optimal range closer to previous studies of $N = 5$ to 12. As circuit size increases, the average post-placement wirelength also increases. This leads to the need for more wire tracks per channel and hence more routing area per tile.

5.1.2 Delay

Figure 10b and Figure 11b show that critical path delay decreases as logic block size increases. This occurs as more nets are implemented using the fast local LB interconnect. Although, the delay through an LB also increases, it does so at a much slower rate. For the range of LB sizes that we investigate, there is no optimal range. Delay continues to decrease as LB size is increased.

5.1.3 CAD Runtime

Logic block size has a significant impact on placement and routing runtime as shown in Figure 10c and 11c. When the packing efficiency of the architecture is not constrained by the number of inputs available to logic blocks, the number of LBs required to implement a circuit is proportional to $1/N$. Since the placement problem size is proportional to the number of LBs to place, we see a $1/N$ trend in placement time. Routing time is also reduced as fewer nets need to be routed on the global routing network.

5.2 Multi-Level Logic Blocks

In this set of experiments, we construct LBs with an effective size ranging from $N_{eff} = N2 \cdot N1 = 2$ to 50. We considered three values for the second level hierarchy size ($N2$=2,4,6) and vary $N1$ accordingly. Inputs at each level were chosen based on Equation 4 using a Rent parameter of p=0.8. We found that the results did not vary significantly across the different values of $N2$. In the following we discuss only the case of $N2 = 4$. The experiments are performed using the Synthetic benchmark suite.

5.2.1 Area

The solid lines in Figure 12 show the logic, routing, and total area results. The dashed lines are the flat architecture results from Figure 10a. This comparison shows that the multi-level architecture is more area-efficient beyond $N_{eff} =$ 20. The are two reasons for this. First, since the overall multi-level local interconnect network is implemented as two stages (second-stage local interconnect and first-stage local interconnect), the MUXes are narrower than in the

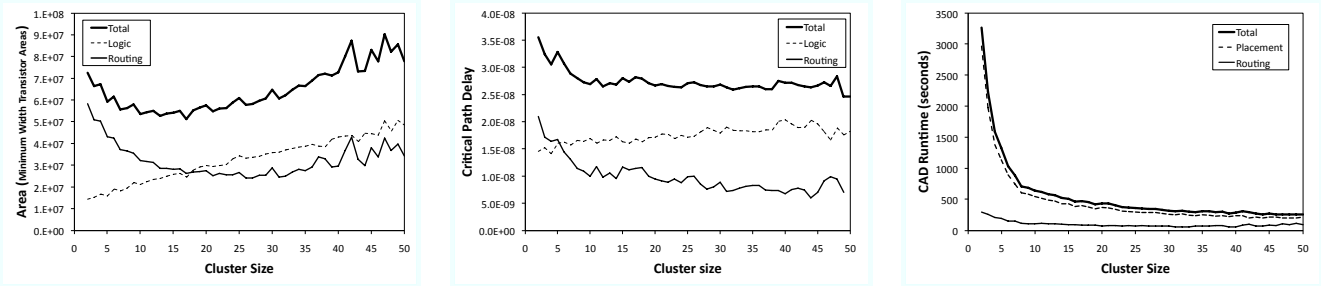

Figure 10: Effect of Flat Cluster Size (using synthetic circuits) on a) Area b) Delay c) CAD Runtime

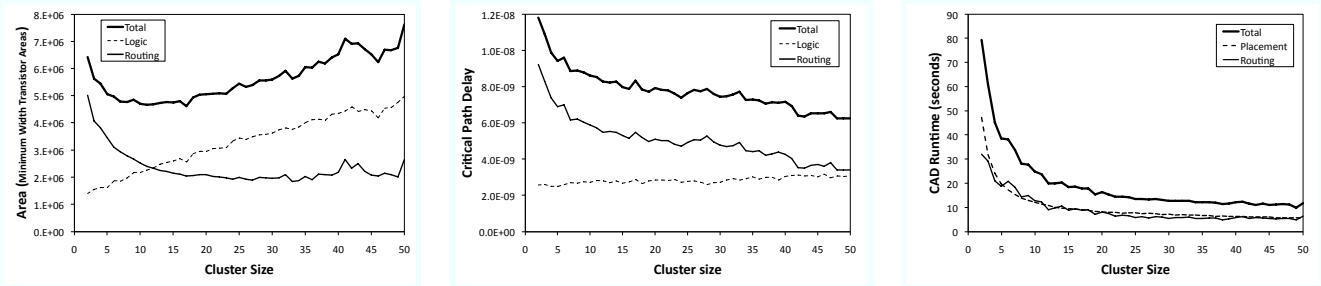

Figure 11: Effect of Flat Cluster Size (using MCNC circuits) on a) Area b) Delay c) CAD Runtime

flat architecture. Thus they need fewer transistors overall to implement.

The second factor is packing efficiency. Both the flat architecture and the multi-level architecture require almost identical number of LBs to implement each circuit. However, we found that the flat architecture was able to absorb more nets, thus requiring the use of fewer LB pins which in turn reduces the routing channel width and routing area. The reason for this is that the multi-level architecture has a physical I/O constraint that must be met at the first-level which in turn limits the size of nets that can be completely absorbed at the second level. This is why the flat architecture uses less routing area up to $N = 30$. Beyond this point, the flat architecture's required channel width begins to increase again which leads to high routing area.

5.2.2 Delay

In Figure 12b, the solid lines show the logic, routing and total critical path delay in the multi-level architecture. We only show the *total* delay from the flat architecture for comparison. The difference in delay is significant. For the same reasons in the area results, we found that packing efficiency played a large role in this gap. Since the multi-level architecture requires more global routing, more of the critical path is implemented using the slower global routing resources.

There is also a secondary factor that leads to a delay overhead in the logic delay of the multi-level architecture compared to the flat architecture. To uncover this, we repeated the experiments on the flat architecture using the same packing and placement solutions from the multi-level experiments. Converting the packing solutions was done by simply *flattening* the netlists produced by the multi-level packing tool. The dashed lines show the results from this approach.

Routing delay remains the same because the sources and sinks of all nets remain at their original locations. Logic delay is slightly worse for the multi-level LBs. In the multi-level architecture, an LB input pin connecting to a BLE input pin must traverse through two levels of local interconnect. Since each level is implemented as 2-stage MUXes, the signal must propagate through a total of four pass transistors compared to only two in the flat architecture. We investigated different MUX implementation schemes (such as one-hot) but found that 2-stage MUXes were still optimal.

5.2.3 CAD Runtime

The behaviour in CAD runtime is the same as that of the flat architecture since the number of LBs, and hence place and route problem sizes, is being varied on the same scale but through a different architecture. We therefore omit the data for brevity.

5.3 Quality versus CAD Runtime Trade-off

Figure 13a and b show the area versus P&R runtime trade-off, and the delay versus P&R runtime trade-off for the flat architecture, respectively. The data points have been normalized to the values for the smallest LB size of $N = 2$. The solid line represents the data for the synthetic benchmark suite and dashed line represents the data for the MCNC20 suite. These figures summarize the CAD runtime trade-offs that can be achieved through logic block architecture size.

In general, there is an asymptote for the amount of CAD runtime reduction that can be achieved. However, it is important to observe that this asymptote is not static. As FPGAs grow in capacity and user circuits become larger, the total runtime increases but the room for improvement through the techniques in the paper also increase. This is highlighted by the gap between the solid line representing

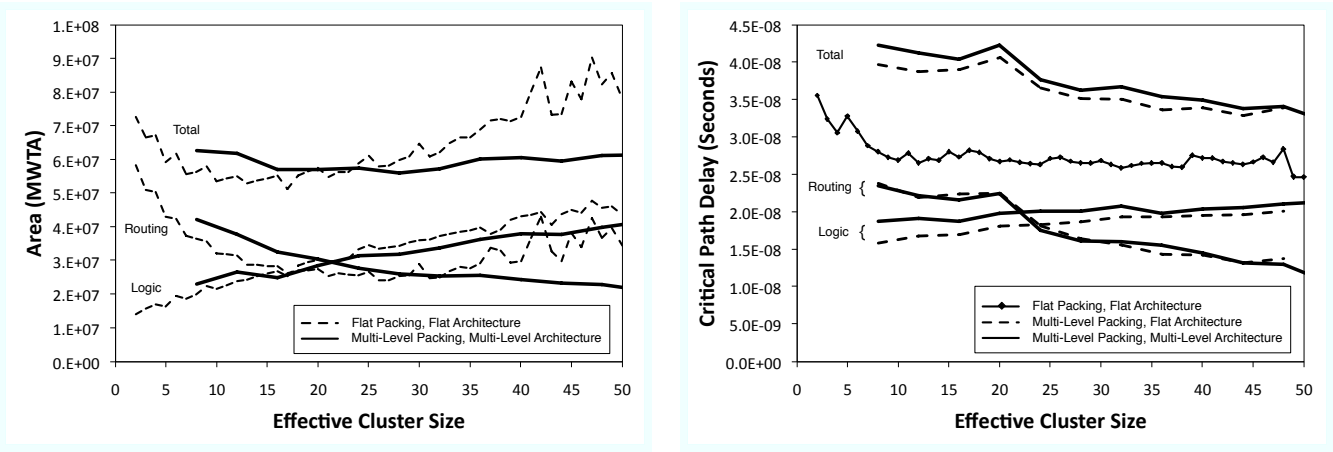

Figure 12: Effect of Multi-Level Cluster Size on a) Area b) Delay

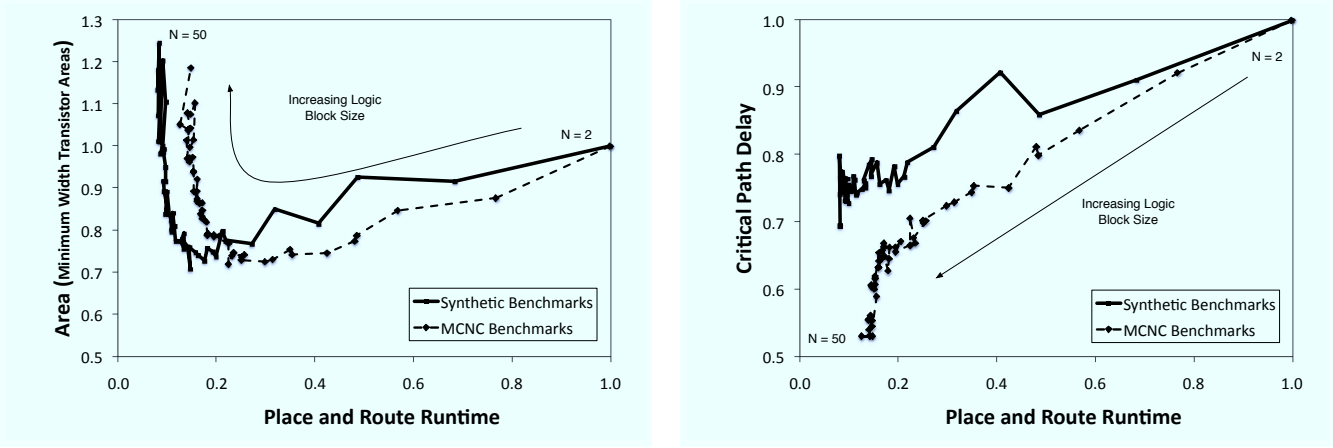

Figure 13: Flat Total Runtime a) Area-Runtime Tradeoff b) Delay - Runtime Tradeoff

the larger synthetic circuits, and the dashed line representing the much smaller MCNC circuits.

These figures also show that the main trade-off for reducing CAD runtime is an increase in area. While we do not see an increase in delay, it is expected that delay would eventually increase at some point (imagine the extreme case, the entire FPGA as a single LB).

Figure 14 shows a similar area and delay versus CAD runtime trade-off for the multi-level architectures. While the trade-off for area is not as steep, the critical path delay that can be achieved by these architectures is worse than the flat architecture.

Overall, there is a clear trade-off between mapping quality (area and delay) and CAD runtime when modifying the logic block architecture.

6. CONCLUSION AND FUTURE WORK

Long runtimes is a growing problem for FPGA CAD. In this paper, we have argued that it is possible to design the architecture of an FPGA while balancing CAD run-time with the more traditional optimization metrics of area, delay, and power. We investigated the use of high-capacity logic blocks to reduce the placement and routing problem sizes directly. We focused on two ways to architect these blocks: direct scaling of a flat logic block and a multi-level approach. We first derived an equation to determine the amount of I/O needed for both types of logic blocks. Then, using an experimental methodology, we quantified the area, delay and place and route runtime trade-offs of these architectures.

We found that the multi-level architecture is slightly more area efficient but slightly less delay efficient when compared to the flat architecture. Using specialized multi-level packing algorithms may help close the quality gap that we observed when using iterative T-VPack. We also found that the I/O constraints of the first hierarchy affects the overall packing efficiency.

Overall, we showed a clear trade-off between mapping quality (area and delay) and CAD runtime that was affected by architecture design. The architectures explored in this paper are relatively straight-forward and interesting solutions may lie in more radical approaches. Runtime-aware architecture design is a very large unexplored area of research. Some interesting directions include:

- Revisit past architecture optimization studies when using only fast CAD tools. Do the architectural conclusions change when only fast CAD tools are used?

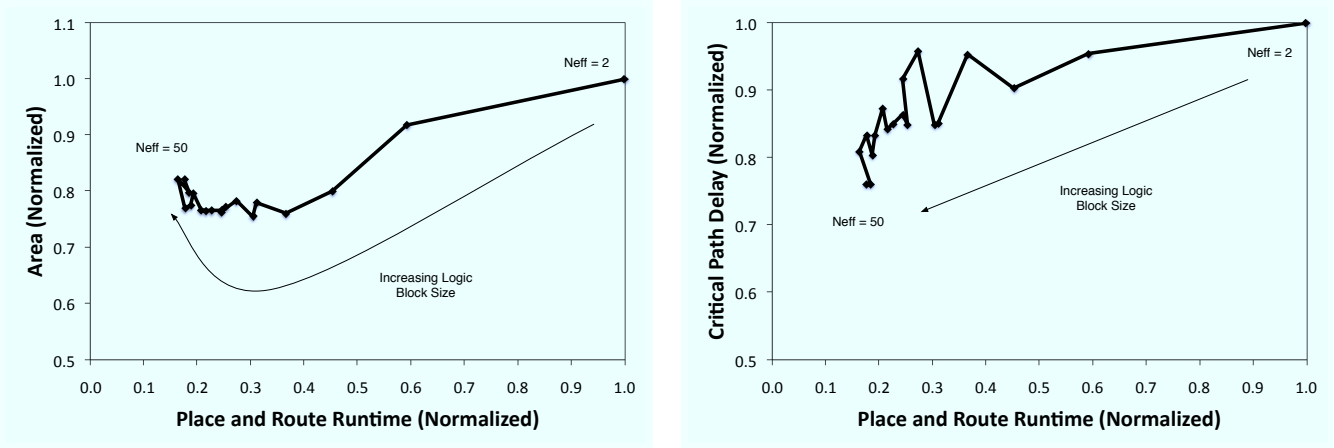

Figure 14: Multi-Level a) Area-Runtime Tradeoff b) Delay - Runtime Tradeoff

- Improving routability is a major goal in all CAD stages prior to routing. Routing itself is also a very timing consuming step in the CAD flow. This paper has focused on logic block architecture. More opportunities for affecting CAD runtime likely exist in the FPGA interconnect design.

- Developing parallel CAD algorithms will no doubt play an important role in the future as desktop CPUs continue to move deeper into the use of multicore architectures. How does the FPGA architecture affect the amount of parallelism that can be employed in the CAD algorithms? Can new architectures be developed that are more amenable to parallel CAD?

Acknowledgment

This research was funded by Altera and the Natural Sciences and Engineering Research Council of Canada.

7. REFERENCES

[1] E. Ahmed and J. Rose. The effect of lut and cluster size on deep-submicron fpga performance and density. *IEEE Trans. on VLSI*, 12(3):288–298, March 2004.

[2] Altera. Apex20 family data sheets.

[3] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, and L. Albertson. Plasma: an fpga for million gate systems. In *Procs. of the Int'l Symp. on Field Programmable Gate Arrays*, pages 10–16, 1996.

[4] V. Betz, J. Rose, and A. Marquardt. Architecture and cad for deep-submicron fpgas. *Springer*, pages ISBN 0–7923–8460–1, 1999.

[5] H. Bian, A. C. Ling, A. Choong, and J. Zhu. Towards scalable placement for fpgas. In *Procs. of the Int'l Symp. on Field Programmable Gate Arrays*, pages 147–156, 2010.

[6] S. Chin and S. Wilton. An analytical model relating fpga architecture and place and route runtime. In *Procs. of the Int'l Conf. on Field-Programmable Logic and Applications*, pages 146–153, Aug. 2009.

[7] J. Cong and Y. Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. In *Proc. of Int'l conf. on Computer-aided design*, pages 48–53, 1992.

[8] J. Cong and M. Romesis. Performance-driven multi-level clustering with application to hierarchical fpga mapping. In *Procs. of the Design Automation Conference*, pages 389–394, 2001.

[9] M. Dehkordi and S. Brown. Performance-driven recursive multi-level clustering. In *Int'l Conf. on Field-Programmable Tech.*, pages 262–269, Dec. 2003.

[10] E. Hung, S. J. E. Wilton, H. Yu, T. C. P. Chau, and P. H. W. Leong. A detailed delay path model for fpgas. In *Procs. of the Int'l Conf. on Field-Programmable Technology*, pages 96–103, Dec. 2009.

[11] M. Hutton. Interconnect prediction for programmable logic devices. In *Int'l workshop on System-level interconnect prediction*, pages 125–131, 2001.

[12] A. Lam, S. Wilton, P. Leong, and W. Luk. An analytical model describing the relationships between logic architecture and fpga density. In *Int'l Conf. on Field-Programmable Logic and Applications*, 2008.

[13] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in fpga interconnect. In *Int'l conf. on Field-Programmable Tech.*, pages 41–48, 2004.

[14] A. Ludwin, V. Betz, and K. Padalia. High-quality, deterministic parallel placement for fpgas on commodity hardware. In *Procs. Int'l Symp. on Field programmable gate arrays*, pages 14–23, 2008.

[15] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose. Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *Proc. of Int'l Symp. on FPGA*, pages 133–142, 2009.

[16] C. Mark, A. Shui, and S. Wilton. A system-level stochastic circuit generator for fpga architecture evaluation. In *Procs. of the Int'l Conf. on Field-Programmable Technology*, 2008.

[17] A. Marquardt, V. Betz, and J. Rose. Speed and area tradeoffs in cluster-based fpga architectures. *IEEE Trans. V. Large Scale Integr. Syst.*, 8(1):84–93, 2000.

[18] C. Mulpuri and S. Hauck. Runtime and quality tradeoffs in fpga placement and routing. In *Int'l Symp on Field programmable gate arrays*, pages 29–36, 2001.

[19] J. Pistorius and M. Hutton. Placement rent exponent calculation methods, temporal behaviour and fpga architecture evaluation. In *workshop on System-level interconnect prediction*, pages 31–38, 2003.

[20] K. K. W. Poon, S. J. E. Wilton, and A. Yan. A detailed power model for field-programmable gate arrays. *ACM Trans. Design Automation of Electronic Systems*, 10(2):279–302, 2005.

[21] D. P. Singh and S. D. Brown. Incremental placement for layout driven optimizations on fpgas. In *Int'l Conf. on Computer-aided design*, pages 752–759, 2002.

[22] R. Tessier. Fast placement approaches for fpgas. *ACM Trans. Design Automation of Electronic Systems*, 7(2):284–305, 2002.

[23] K. Vorwerk, A. Kennings, and J. W. Greene. Improving simulated annealing-based fpga placement with directed moves. *Trans. Computer-Aided Design of Integrated Circuits and Systems*, 28(2):179–192, 2009.

[24] A. G. Ye. Using the minimum set of input combinations to minimize the area of local routing networks in logic clusters containing logically equivalent i/os in fpgas. *IEEE Trans. Very Large Scale Integr. Syst.*, 18(1):95–107, 2010.

[25] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm early design exploration. *IEEE Trans. Electron Devices*, 53(11):2816–2823, 2006.