# Rethinking FPGAs: Elude the Flexibility Excess of LUTs with And-Inverter Cones

Hadi Parandeh-Afshar
hadi.parandehafshar@epfl.ch

Hind Benbihi
hind.benbihi@epfl.ch

David Novo
david.novobruna@epfl.ch

Paolo Ienne
paolo.ienne@epfl.ch

Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland

## ABSTRACT

*Look-Up Tables (LUTs)* are universally used in FPGAs as the elementary logic blocks. They can implement any logic function and thus covering a circuit is a relatively straightforward problem. Naturally, flexibility comes at a price, and increasing the number of LUT inputs to cover larger parts of a circuit has an exponential cost in the LUT complexity. Hence, rarely LUTs with more than 4–6 inputs have been used. In this paper we argue that other elementary logic blocks can provide a better compromise between hardware complexity, flexibility, delay, and input and output counts. Inspired by recent trends in synthesis and verification, we explore blocks based on *And-Inverter Graphs (AIGs)*: they have a complexity which is only linear in the number of inputs, they sport the potential for multiple independent outputs, and the delay is only logarithmic in the number of inputs. Of course, these new blocks are extremely less flexible than LUTs; yet, we show (i) that effective mapping algorithms exist, (ii) that, due to their simplicity, poor utilization is less of an issue than with LUTs, and (iii) that a few LUTs can still be used in extreme unfortunate cases. We show first results indicating that this new logic block *combined* to some LUTs in hybrid FPGAs can reduce delay up to 22–32% and area by some 16% on average. Yet, we explored only a few design points and we think that these results could still be improved by a more systematic exploration.

## Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles—*Logic arrays, Combinational logic*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

## General Terms

Design, Performance


*FPGA'12,* February 22–24, 2012, Monterey, California, USA.


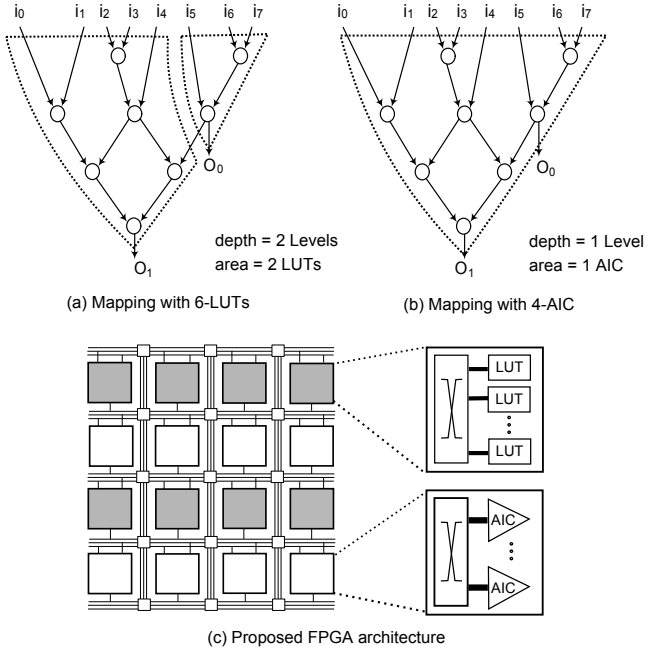


**Figure 1: Flexibility, bandwidth, cost, and delay. (a)–(b) *And-Inverter Cones (AICs)* can map circuits more efficiently than LUTs, because AICs are multi-output blocks and cover more logic depth due to their higher input bandwidth. (c) A possible integration of AIC clusters in an FPGA architecture.**

## Keywords

FPGA Logic Block, Logic Synthesis, And-Inverter Graph, And-Inverter Cone

## 1. INTRODUCTION

Since their commercial introduction in the '80s, FPGAs have been essentially based on *Look-Up Tables (LUTs)*. $K$-input LUTs have one great virtue: they are generic blocks which can implement any logic function of $K$ inputs, and this makes it relatively easy to perform at least some elementary technology mapping: crudely, the problem of mapping reduces to cover the circuit to map with $K$-input subgraphs, irrespective of the function they represent. This flexibil-

ity, and the consequent advantages, do not come for free: LUTs tend to be large (roughly, their area grows exponentially with the number of inputs) and somehow slow (equally roughly, the delay grows linearly with the number of inputs). Also, the number of outputs is intrinsically one and internal fan-out in the subgraphs used for covering is not really possible. Fig. 1(a) suggests graphically how the small number of inputs and the absence of intermediate outputs limit the usefulness of LUTs.

This seems to suggest that perhaps it would be wise to look for less versatile but more efficient logic blocks. In fact, researchers have at times looked into alternate blocks ever since FPGAs have attracted growing research and commercial interest. Yet, naturally, these alternate structures have been somehow related to the logic synthesis capabilities of the time, and thus have almost universally addressed programmable AND/OR configurations in the form of small *Programmable Array Logics (PALs)* (e.g., [14, 13, 8]). Traditionally, synthesis has been built on the sum of products representation and on algebraic transformations, but new paradigms have emerged in recent years. The one we are interested in is based on *And-Inverter Graphs (AIGs)* as implemented in the well-know academic synthesis and verification framework ABC [20]. This representation, in which all nodes are 2-input AND gates with an optional inversion at the output, is not new [11], but has received interest in recent years due to some fortunate combination when used with, for instance, *Boolean satisfiability (SAT)* solvers. Once a circuit is written and optimized in the form of an AIG, one can find very many AIG subgraphs of various depth rooted at different nodes in the circuit.

Thus, we introduce a new logic block that we call *And-Inverter Cone (AIC)*. An AIC (which is explained in detail in Fig. 3) is essentially the simplest reconfigurable circuit where arbitrary AIGs can be naturally mapped: it is a binary tree composed of AND gates with a programmable conditional inversion and a number of intermediary outputs. Compared to LUTs, AICs can be richer in terms of input and output bandwidth, because their area grows only linearly with the number of inputs. Also their delay grows only logarithimically with the input count and intermediate outputs are easier to implement. This makes it possible for AICs to cover AIG nodes more efficiently, as suggested in Fig. 1(a)-(b). In this paper, we will explore the value of AICs both as the sole components of new FPGAs as well as logic blocks for some hybrid FPGA made of both LUTs and AICs, as illustrated in Fig. 1(c). Although far from exploring comprehensively the space of AIC-based solutions, our results suggest that some hybrid solutions look particularly promising and, at the very least, deserve some further attention to refine our analysis.

The rest of the paper adapts the traditional CAD flow used on conventional FPGAs to the needs of AICs and, simultaneously, uses some of the partial results to fix the structure of our novel FPGA. Fig. 2 suggests this graphically: Section 2 addresses the design of the AIC to suit the abilities of modern AIG synthesis. Section 3 adapts traditional technology mapping to the new block. Section 4 looks at how to combine logic blocks in larger clusters with local routing, and Section 5 discusses the packing problem to complete the flow. Sections 6 and 7 then report our experimental results. We discuss related work in Section 8 and then wrap up with some conclusive remarks.

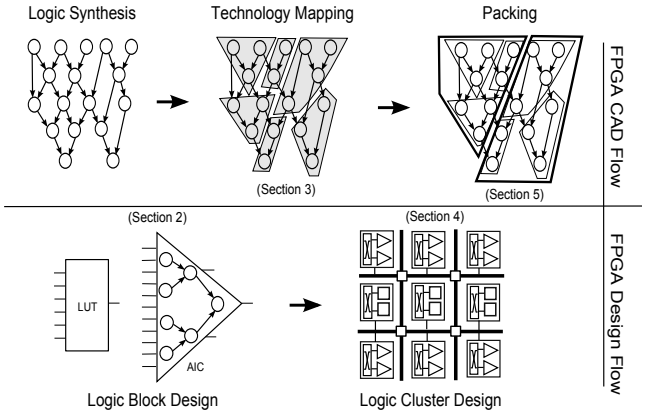


**Figure 2: The paths to design and use a novel FPGA with AICs. In this paper, we alternate between adapting the traditional CAD flow to our new needs and using the results to fix our architecture. To each of the last four steps is devoted one of the sections of the paper, as indicated.**

# 2. LOGIC BLOCK DESIGN

A new logic block is proposed in this section. This attempts to reduce the degree of generality provided by typical LUTs in order to obtain faster mappings. Unlike LUTs, our logic block is not able to implement all possible functions of its inputs. In the following, the choice of logic block is motivated and its architecture is discussed.

## 2.1 An AIG-inspired logic block

An *And-Inverter Graph (AIG)* is a *Directed Acyclic Graph (DAG)*, in which the logic nodes are two-input AND gates and the edges can be complemented to represent inverters at the node outputs. AIGs have been proven to be advantageous for combinational logic synthesis and optimization [20]. This graph representation format is also used for technology mapping step in both FPGA and ASIC designs [4].

Interestingly, AIGs include various cone-like subgraphs rooted at each node with different depths. Usually, the subgraphs with lower depths are more symmetric and resemble full binary trees. The frequent occurrence of such conic subgraphs serves as motivation of this work, where we propose a new logic block that can map cones with different depths more efficiently than LUTs. The basic idea is to have a symmetric and conic block with depth $D$, which maps arbitrary AIG subgraphs with depth $\leq D$. This logic block is called *And-Inverter Cone (AIC)*.

To illustrate the potential benefits of AICs with respect to LUTs, we refer to Fig. 1, where two levels of LUTs are required to map the same functionality that can be mapped onto a single AIC. The reason for that is twofold: on the one hand, the LUT size is limited to six inputs and the entire AIG (8 inputs) can not fit into just one 6-LUT. On the other hand, even if the size of the LUT was big enough, the mapping would still use two LUTs, as the AIG has two distinct outputs. It is worth mentioning that increasing the LUT size to accommodate more inputs would result in a huge area overhead. Instead, the proposed AIC inherently offers smaller area and propagation delay than a LUT for the same number of inputs. For example, a 4-AIC with 16 in-

| Block | inputs | outputs | 2:1 mux | config bits |
|---|---|---|---|---|
| 2-AIC | 4 | 1 | 3 | 3 |
| 3-AIC | 8 | 3 | 7 | 7 |
| 4-AIC | 16 | 7 | 15 | 15 |
| 5-AIC | 32 | 15 | 31 | 31 |
| 6-AIC | 64 | 31 | 63 | 63 |
| 6-LUT | 6 | 1 | 64 | 64 |

**Table 1: AICs have less configuration bits than LUTs, while they can implement circuits with a much greater number of inputs (e.g., a 6-AIC includes 8 times more inputs than a typical 6-LUT).**

puts requires half the area of a 6-input LUT—using the area model of Section 6.1 with less delay. Clearly, the fact that more wires need to be connected to the AICs creates new routing congestion issues. However, as detailed in Section 4, these can largely be alleviated by packing several AICs in a limited bandwidth AIC cluster with local interconnect.

## 2.2 AND-Inverter Cone (AIC) Architecture

Fig. 3 shows the architecture of an *And-Inverter Cone (AIC)*, which has five levels of cells. Each cell can be configured as either a two-input NAND or AND gate. Notice that each cell has an AIC output, except for the cells belonging to the lowest level of the AIC. This provides access to intermediate nodes as in the example of Fig. 1. Moreover, these outputs enable to configure a bigger AIC as multiple smaller ones. For example, the AIC of Fig. 3, implements the AIG of Fig. 1 at the right-hand side while the left-hand side can be used to implement other functions with various combinations of 2-, 3-, and 4-AICs. Accordingly, a 5-AIC contains two 4-AICs, four 3-AICs, or eight 2-AICs.

Generalizing, each $D$-AIC has $2^D - 1$ cells, $2^D$ inputs and $2^D - 2^{D-1} - 1$ outputs. In the rest of the paper, we consider $D$-AICs with depths from three to six, and we will study the effect of the allowed AIC depth on the mapping solution. Depths greater than six are not considered, as they require a huge input bandwidth, which may result in major modifications of the global routing network of current FPGAs. Table 1 compares different $D$-AICs with the conventional 6-LUT in terms of IO bandwidth, number of configuration bits and multiplexers.

# 3. TECHNOLOGY MAPPING

During technology mapping, the nodes comprising the AIG are clustered into subgraphs that can be mapped onto an AIC or a LUT. This can be done in multiple ways depending on the optimization objectives including delay and area.

In this work, the primary optimization objective of technology mapping is delay minimization and consequently a mapping solution is said to be *optimal* if the mapping delay is minimum. Area reduction is also considered but just as a secondary optimization objective. Technology mapping for AICs is similar to the typical LUT technology mapping but adapted to the peculiarities of AICs, such as the fact that multiple outputs are possible. In the rest of the section, the mapping problem is first formalized and then the main four steps of the mapping algorithm are described in detail.

## 3.1 Definitions and Problem Formulation

A technology independent synthesized netlist (AIG format) is input to our mapping heuristic. Such netlist is automatically produced by ABC [20]. We take the input netlist and extract the combinational parts of the circuit and represent them by a DAG $G = (V(G), E(G))$. A node $v \in V(G)$ can represent an AND gate, a primary input (PI), a pseudo input (PSI, output of a flipflop), a primary output (PO), or a pseudo output (PSO, input of a flipflop). A directed edge $e \in E(G)$ represents an interconnection wire in the input netlist. The edge can have the *complemented* attribute to represent the inversion of the signal.

At a node $v$, the depth $depth(v)$ denotes the length of the longest path from any of the PIs or PSIs to $v$. The height $height(v)$ denotes the the length of the longest path from $v$ to any of the POs or PSOs. Accordingly, the depth of a PI or PSI node and the height of a PO or PSO node are zero.

The mapping algorithm that we use in this work is a modified version of the classical depth-optimal LUT mapping algorithm [6]. It is well known that the problem of minimizing the depth can be solved optimally in polynomial time using dynamic programming [6, 15]. However, we also target area-minimization as a secondary objective, which is known to be NP-hard for LUTs of size three and greater [7, 16]. We use *area flow* heuristic [19] for area approximation during the mapping.

The mapping of a graph in LUTs requires different considerations. For a node $v$, there exist several subgraphs containing $v$ as the root, which are called *cones*. Accordingly, $C_v$ is a cone that includes node $v$ in its root and some or all of its predecessors. For mapping $C_v$ by a LUT, it should be $K$-feasible, where $inputs(C_v) \leq K$. Moreover, the cone should be *fanout-free*, meaning that the only path out of $C_v$ is through $v$. If the cone is not fanout free, then the node which provides the fanout may be duplicated and will be mapped by other LUT(s), as the primary minimization objective is depth.

The AICs mapping cone candidates of $v$ are extracted differently. In this case, rather than being $K$-feasible, a cone $C_v$, to be mappable on a $D$-AIC block, should be depth feasible, where $depth(C_v) \leq D$. The other constraint is that the nodes at lowest depth of $C_v$, should not have any path to a node outside $C_v$, otherwise such nodes are removed from $C_v$. This condition ensures that $C_v$ to be mappable to an AIC such as the one illustrated in Fig. 3, in which no AIC output is driven by the nodes at the lowest level of the AIC.

When AICs are considered as the mapping target in addition to LUTs, the definition of the problem of mapping for depth does not change. The only difference is that the cone candidates of AICs are added to the cone candidates of LUTs for each node in the graph. Although the conditions of eligibility for LUTs and AICs are different, it is possible to have common cones between the two that are treated as separate candidates.

Next, the main steps of the mapping algorithm are described in detail.

## 3.2 Generating All Cones

To generate all $K$-feasible cones, we use the algorithm described in [9, 22], in which the cones of a node are computed by combining the cones of the input nodes in every possible way. This step of the mapping takes a significant portion of

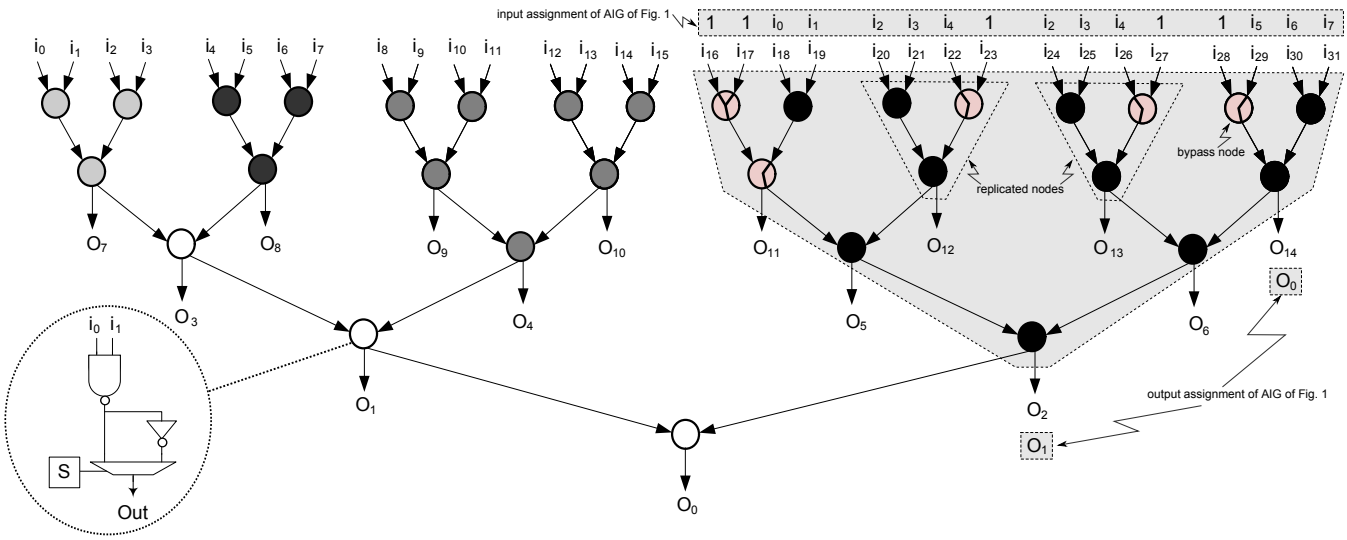


**Figure 3: Architecture of 5-AIC (AND-Inverter Cone), which has five levels of cells that are programmable to either AND or NAND gates. The 5-AIC can also be configured to 2-, 3-, and 4-AICs in many ways (highlighted cells show one possibility), without any need for extra hardware. The AIG of Fig. 1 is mapped onto the right-hand side. To propagate a signal, we can configure a cell to the bypass mode (e.g., forcing one input to 1 when this is operated as an AND). Moreover, some AIG nodes need to be replicated when the fanout of an internal value is larger than one.**

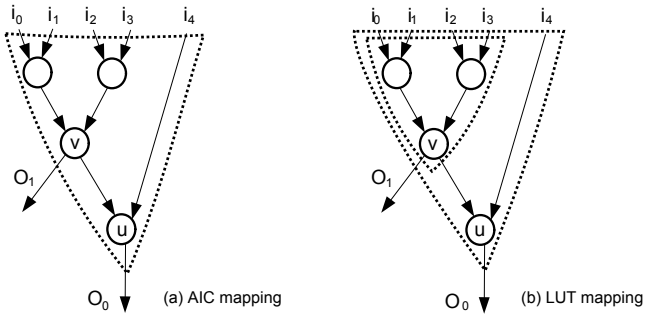


**Figure 4: Difference between LUT and AIC mapping. Since AICs are inherently multi-output blocks, the same cone rooted at $u$ in (a) can also be a (free) mapping cone of $v$, while in LUT mapping, no common cone exist for any two nodes (b).**

the total execution time, specially when $K$ is a large value such as six.

The cone generation for AICs is different from the cone generation for LUTs, as the cones of each node are produced independently from the cones of its input nodes. To generate all possible $D$-AIC mappable cones for a node $v$, the subgraphs rooted at $v$ are examined by varying the cone depth from two to $D$. All possible subgraphs that meet the AIC mapping conditions described in section 3.1, are added to the cone set of node $v$. If a cone $C_v$ satisfies the depth condition, but has a fanout node $u$ at the lowest depth of the cone, $u$ will be removed from $C_v$; if this still satisfies the depth condition, the cone will be added to the $D$-AIC mappable cone set.

The main difference between the cone generation for AICs and LUTs is having common cone candidates for different nodes, as shown in Fig. 4. This is possible, as AICs are multi-output. In this figure, the cone that has $u$ as its root, can be used to map both $v$ and $u$. Therefore, this cone should be in the AIC cone sets of both nodes. We call this cone as a *free* cone for node $v$, as it maps $v$ for free when it is selected for $u$ mapping.

The time complexity of the $D$-AIC cone generation is $O(M \cdot D)$, where $M$ is the number of nodes in the graph and $D$ is the maximum depth of an AIC block.

## 3.3 Forward Traversal

Once the cones sets of both LUTs and AICS are computed for every node in the graph, the next step is to find the best cone of each node by traversing the graph in topological order. Since the primary objective in this work is to minimize the depth, the best cone of node $v$ is the one that gives $v$ the lowest depth. If there is more than one option, the cone which brings less area flow to $v$ is selected (see [19] for further details). The depth and area flow of $v$, when mapped onto cone $C_v$, are dependent on the depth and area flow values of the $C_v$ input nodes.

To compute the depth and area flow of node $v$, we use Equations 1 and 2, respectively. Since the FPGA blocks, including $K$-LUTs and $D$-AICs, are heterogeneous and have different depths, we should consider the interconnection wire delays for the depth computation of each node, similar to the edge-delay model [23]. Although we have both local (intra cluster) and global (inter cluster) routing wires, which have different delays, we assume that all wires have unique delay equal to the average delay of the local and global wires.

$$dp(v) = \max(dp(In(C_v)) + dp(C_v) + dp(wire)) \quad (1)$$

$$af(v) = \sum_{i=0}^{nIn(C_v)} (af(In_i(C_v)) + area(C_v) \quad (2)$$

**Algorithm 1** Find the best cone for each node of the DAG

```
1: BestC_v.dp ← ∞
2: BestC_v.af ← ∞
3: for i = 1 → nC_v(LUT) do
4:    v.setdp(C_v(i))
5:    v.setaf(C_v(i))
6:    cond_1 ← C_v(i).dp < BestC_v.dp
7:    cond_2 ← C_v(i).dp = BestC_v.dp
8:    cond_3 ← C_v(i).af < BestC_v.af
9:    if cond_1 || (cond_2 && cond_3) then
10:      BestC_v ← C_v(i)
11:   end if
12: end for
13: for i = 1 → nC_v(AIC) do
14:   v.setdp(C_v(i))
15:   v.setaf(C_v(i))
16:   cond_1 ← C_v(i).dp < BestC_v.dp
17:   cond_2 ← C_v(i).dp = BestC_v.dp
18:   cond_3 ← C_v(i).af < BestC_v.af
19:   if cond_1 || (cond_2 && cond_3) then
20:      BestC_v ← C_v(i)
21:   end if
22:   cond_1 ← C_v(i).dp < BestBackupC_v.dp
23:   cond_2 ← C_v(i).dp = BestBackupC_v.dp
24:   cond_3 ← C_v(i).af < BestBackupC_v.af
25:   if C_v(i).root = v then
26:      if cond_1 || (cond_2 && cond_3) then
27:         BestBackupC_v ← C_v(i)
28:      end if
29:   end if
30: end for
```

In the above equations, $dp(C_v)$ and $area(C_v)$ are the depth and area of the logic block that $C_v$ can be mapped on. This block can be either a $K$-LUT or a $D$-AIC. If $C_v$ is a *free* cone of node $v$, then $dp(C_v)$ and $dp(In(C_v))$ will refer to the depth and inputs of the sub-AIC in $C_v$. And for area flow computation, the term $area(C_v)$ will be removed from Equation 2.

Algorithm 1 presents the pseudo-code of the algorithm used to find the best cone of each AIG node. This function iterates over all generated cones for both LUTs and AICs of node $v$ to find the best cone that has the lowest depth. If two cones have the same depth, the one that requires smaller area is selected. If the best cone of node $v$ is a *free* cone, this cone will be selected for the mapping, if and only if the root of the cone—which is not $v$—is visible in the final mapping solution and this cone is the best cone of the root node as well. If one of these two conditions does not hold, then we need to select another cone as the best cone for $v$. Therefore, it is essential to maintain a non-*free* best cone—$v$ is the root of such a cone—for $v$ as a backup best cone.

## 3.4 Backward Traversal

In this step, the graph is covered by the best cones of the visible nodes in the graph, which are added to the mapping solution set $S$. A node is called *visible*, if it is an output or input node of a selected cone in the final mapping. Initially POs and PSOs are the only visible nodes and $S$ is empty. The graph traversal is performed in reverse topological order from POs and PSOs to PIs and PSIs. If the visited node $v$ is visible, then its best cone, $BC_v$, is selected for the mapping and is the added to $S$. Then, all the input nodes of $BC_v$ become visible and the graph traversal continues. If the $BC_v$ is a *free* cone and it is already in $S$, there is no need to add it again and only the heights of the input nodes of $v$ are updated. Otherwise, if the *free* cone is not in $S$, then the backup $BC_v$, which has $v$ as its root, is selected for mapping and is added to $S$. During the backward traversal, the height of each visible node is updated. Once a $BC_v$ is selected for mapping, the height of its input nodes are updated by adding the height of $v$ to the depth of $v$ within the target AIC or LUT.

## 3.5 Converting Cones to LUTs and AICs

The mapping solution $S$, which is generated during the Backward Traversal, includes all the cones that cover the graph. The next step is mapping the cones in $S$ to either a $K$-LUT or a $D$-AIC. If the selected cone belongs to the $K$-feasible cone set of node $v$, then it should be implemented by a LUT. Otherwise, the cone is a $D$-AIC mappable cone, which is implemented by an AIC. The depth of the cone defines the type of the target AIC block.

# 4. LOGIC CLUSTER DESIGN

The proposed AICs require a much higher IO bandwidth than typical LUTs. In order to alleviate the routing problem that may result from that increase, we propose to group multiple AICs into an AIC cluster with local interconnect.

To form an AIC cluster, we integrate $N$ $D$-AICs, optional flipflops at the outputs of $D$-AICs to support sequential circuits, and an input and an output crossbar. The input crossbar drives the inputs of the AICs in the cluster, and the output crossbar drives the outputs of the cluster. Since we do not want to change the inter-cluster routing architecture of the FPGAs, we use the same bandwidth of LUT-based clusters for AIC clusters and keep the AIC cluster area close to the area of the reference LUT cluster, which is the *Logic Array Block (LAB)* in the *Altera Stratix-III*.

To study the effect of the AIC size on the mapping results, we select different $D$-AICs as the base logic block in a cluster, where $D$ varies from three to six and can be configured to implement the AIC blocks that have $depth \leq D$. However, the number of the $D$-AIC blocks in the cluster, $N$, varies for different $D$ values such that the number of sub-AICs in the cluster remains the same and no changes occur in the cluster crossbars.

The two crossbars in the AIC cluster are the main contributors to the cluster area. Crossbars are basically constructed with multiplexers and their area depends on their density and on the number of the crossbar inputs and outputs. Since both crossbars get the outputs of $N$ $D$-AICs as the input, reducing the number of the $D$-AIC outputs will significantly reduce the area share of the crossbars. Originally, each $D$-AIC has $2^D - 2^{D-1} - 1$ outputs, but in our experiments, we observed that in the extreme case only $2^{D-2}$ outputs are utilized and that is when a $D$-AIC is configured to $2^{D-2}$ 2-AICs. Hence, a very simple sparse crossbar is added at the output of each $D$-AIC to reduce the number of $D$-AIC outputs to $2^{D-2}$.

The second technique used to reduce the crossbar area is to decrease its connectivity and make it sparse. To trade-off the crossbar density and packing efficiency in the AIC cluster, we measured the packing efficiency of the clusters having an input crossbar with 50%, 75%, and 100% connec-

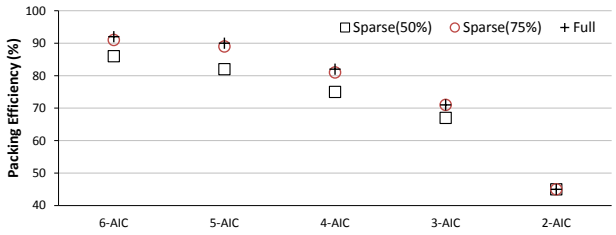


**Figure 5: The packing efficiency of three crossbar connectivity scenarios: 50%, 75%, and 100%. The allowed cone depth in technology mapping is varied to study the effect of AIC size on the packing quality.**

tivities. The packing efficiency is the ratio of the number of AIC clusters, assuming that each AIC cluster has unlimited bandwidth and the actual number of AIC clusters that is obtained from packing. To calculate the number of clusters in the ideal packing, we use Equation 3. In this equation, $nC_i$ is the number of cones with depth $i$. Fig. 5 shows the results of this experiment for different base AIC blocks in the cluster. The reported efficiency is the average packing efficiency of the 20 biggest MCNC benchmarks.

$$nClusters_{ideal} = \sum_{i=2}^{6} \left(\frac{nC_i}{N \cdot 2^{6-i}}\right) \quad (3)$$

One observation from Fig. 5 is that the packing efficiency is substantially reduced for all the three scenarios, when the allowed cone depth in the technology mapping is reduced. This is reasonable, as the probability of input sharing and open inputs is reduced for smaller cones. Moreover, when smaller AICs are packed to a $D$-AIC, a larger number of the $D$-AIC outputs are utilized, which increases the output bandwidth requirement. The second observation is that reducing the crossbar connectivity to 75% largely maintains the packing efficiency of the full crossbar. However, the packing efficiency for the crossbar with 50% connectivity decreases to a larger extent. Therefore, one option to reduce the crossbar area without having a sensible degradation in packing efficiency is to set the crossbar connectivity to 75%.

Exploiting the mentioned crossbar simplifications, and by using the area model of Section 6.1, the area of the AIC cluster remains close to the area of a LAB, when three 6-AICs, six 5-AICs, twelve 4-AICs, or twenty four 3-AICs are integrated in the AIC cluster. As mentioned, the input/output crossbars of the AIC cluster are fixed for all scenarios.

# 5. PACKING APPROACH

In the previous section, we defined the architecture of the AIC cluster. Given the AIC and LUT clusters, the next step is to pack the technology mapped netlist onto the clusters. For the packing, we use the *AAPack* [18] tool, which is an architecture-aware packing tool developed for FPGAs. The input to *AAPack* is the technology mapped netlist with unpacked blocks, as well as a description of an FPGA architecture. The output is a netlist of *packed* complex blocks that is functionally equivalent to the input netlist. Similarly, we also use *AAPack* to pack LUTs in LABs.

The packing algorithm uses an affinity metric to optimize the packing. This affinity metric defines the amount of net sharing between $p$, which is a packing candidate, and $B$, which is a partially filled complex block. In the architecture file, the complex block should be represented as an ordered tree. Nodes in the tree correspond to physical blocks or modes. The root of tree corresponds to an entire complex block and the leaf nodes correspond to the primitives within the complex block. For the $D$-AIC complex block, we construct a tree similar to the DSP block multiplier tree in the original paper, by which we define different configuration modes of the $D$-AIC. The number of AICs in the cluster as well as the crossbars structure are also defined in the architecture file. The information is used by the packer to group the individual blocks in clusters. During the packing process, some routability checking are performed to ensure (local and global) routability of the packing solution, which considers the intra-block and the general FPGA interconnect resources.

| **Component** | **Area ($\mathrm{Tr_{minW}}$)** |
|---|---|
| 6-AIC block | 1,512 |
| 6-AIC output Xbar | 217 |
| 6-AIC FFs and muxes | 1,104 |
| AIC cluster input Xbar | 22,072 |
| AIC cluster out Xbar | 2,660 |
| AIC cluster buffers | 1,447 |
| **AIC cluster with three 6-AICs** | **34,678** |
| ALM | 1,751 |
| LAB in Xbar | 16,251 |
| LAB buffers | 470 |
| **LAB with ten ALMs** | **34,231** |

**Table 2: Areas of different components in an AIC cluster and in a LAB, measured in units of minimum-width transistor area.**

# 6. EXPERIMENTAL METHODOLOGY

In this work, we use a classic area and delay model [5]: The area model is based on the transistor area in units of minimum-width transistor area; the rationale is that to a large extent the total area is determined by the transistors more than by the metal connections. For the delay model, circuits are modeled using SPICE simulations for 90-nm CMOS process technology.

## 6.1 Area Model

The area modeling method requires a detailed transistor-level circuit design of all the circuitry in the FPGA [5]. Fig. 6 shows an AIC cluster with three 6-AICs. Table 2 lists the area of different components in the AIC cluster and in a LAB in terms of number of minimum-width transistors. *ALM* stands for *Adaptive Logic Module*, which is the logic block in *Altera Stratix-II* and in following series. Based on this table, the area of an AIC cluster with three 6-AICs and the crossbars mentioned in Section 4 is marginally larger than a LAB with 10 ALMs. As mentioned in Section 4, the AIC cluster has almost the same area when the basic AIC block is changed.

## 6.2 Delay Model

The circuit level design of the AIC cluster suggested in Fig. 6 is also used for accurate modeling of the cluster delays.

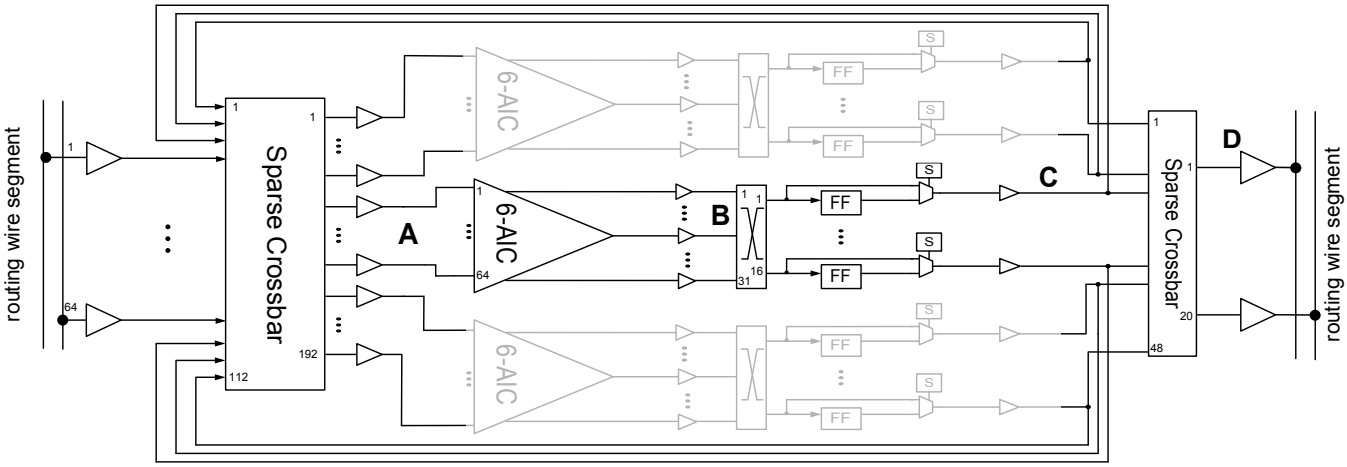


**Figure 6: Structure and delay paths of an AIC cluster with three 6-AICs.**

| Path | Description | Delay (ps) |
|---|---|---|
| A → B | 6-AIC main output | 496 |
| B → C | crossbar and FF-Mux | 75 |
| C → D | output crossbar of cluster | 50 |

**Table 3: Delays of different of paths in the AIC cluster of Fig. 6.**

The crossbars in this figure are developed using multiplexers, and for these we adopted the two level hybrid multiplexer that is used in Stratix-II [17]. Hence, the critical path of each crossbar goes through two pass-gates, with buffers on the inputs and outputs of the components that include pass transistors.

We performed SPICE simulations with 90nm 1.2 V CMOS process, to determine the delay of all paths in the cluster shown in Fig. 6. The results are listed in Table 3. For the path between B and C, the delay number relates to the path that goes through the main output of the 6-AIC, which has the longest path. These delay numbers are used in the technology mapping to minimize the delay of the mapped circuit.

We also measured the delay of a LAB by SPICE simulation. Simulation results revealed that the delay of a 6-LUT in an ALM, excluding the LAB input crossbar, in 90nm CMOS process, is between 280ps and 500ps, taking into account that different LUT inputs have different delays. We use the average delay (390ps) for our experiments. Based on [1], the 6-LUT delay in 90nm process technology has a delay between 162ps to 378ps and considering the extra multiplexers that exist on the LUT output path in the ALM structure, our delay numbers appear realistic.

# 7. RESULTS

We contrast three architectures and various mapping strategies, using the MCNC benchmarks [24]. We consider the original FPGA, a homogeneous FPGA exclusively composed of AIC clusters, and a hybrid FPGA composed of both LUTs and AIC clusters as different experiment scenarios. In the hybrid structure, we also vary the base AIC block of the AIC-cluster from 3-AIC to 6-AIC.

| Mapping Scenario | Intra-cluster Wires |
|---|---|
| LUT | 50% |
| 6-AIC | 34% |
| LUT/6-AIC | 35% |
| LUT/5-AIC | 37% |
| LUT/4-AIC | 38% |
| LUT/3-AIC | 40% |

**Table 4: Average ratio of intra cluster wires for the different mapping scenarios.**

Fig. 7 shows the logic delays of the benchmarks for the mentioned scenarios. The main observation is that the lowest logic delay relates to the hybrid structure, as we have both LUTs and AICs mapping options. Moreover, except for the *ex5p* and *frisc* benchmarks, the logic delay is always reduced when deeper cones are allowed, which appears predictable as a general trend. This is also visible in the number of logic-block levels on the critical path, either LUTs or AICs, as shown in Fig. 8; the graph gives an indication of the routing wires necessary to connect the logic blocks of the circuits: although some logic delays are higher for deeper cones, their total delay can be still better due to the reduced number of wires between logic blocks. Comparing LUT-only and AIC-only implementations, we see that there are circuits that have better logic delay when LUTs are used, but on average AIC-only implementation has 28% less logic delay. Moreover, except for *tseng* and *des*, the number of logic blocks on the critical path (and thus routing wires) in the AIC-only implementation is less than or equal to that of the LUT-only one.

As the current release of VPR 6.0 does not support timing driven placement and routing, we set a fixed delay value for the interconnecting wires in order to estimate the total circuit delay. This delay number is different for the different mapping scenarios and its value is specified based on the delay and used ratio of intra and inter cluster wires for each mapping scenario that is reported in Table 4. Using this wire delay, we compute the routing delay of the critical path of the circuits, using the number of logic blocks in these paths. Fig. 9 illustrates a rough estimation of the

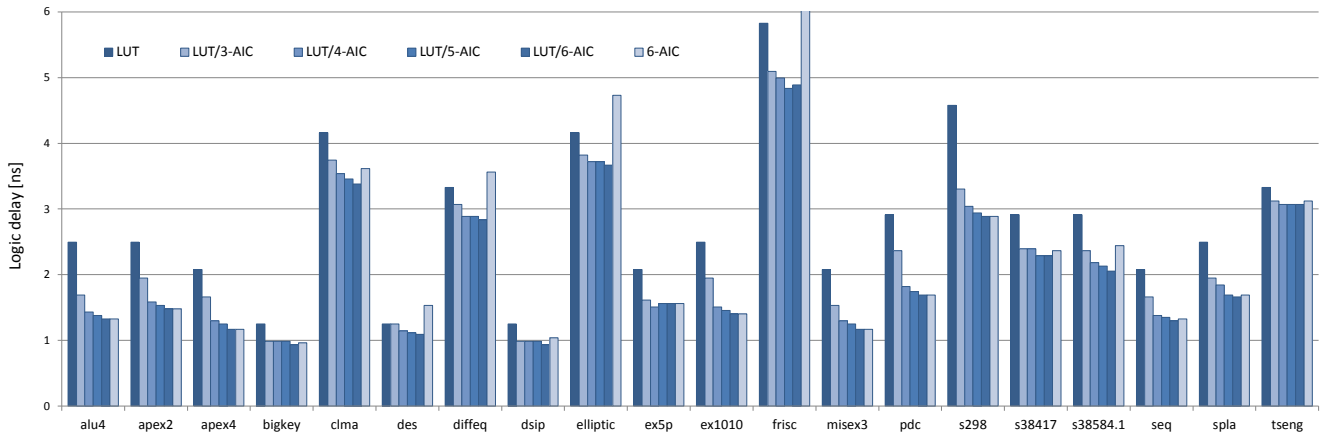


**Figure 7: Logic delay of all benchmarks in the original FPGA (LUT), for the FPGA composed only of AIC (6-AIC), and for a hybrid FPGA (LUT/6-AIC).**

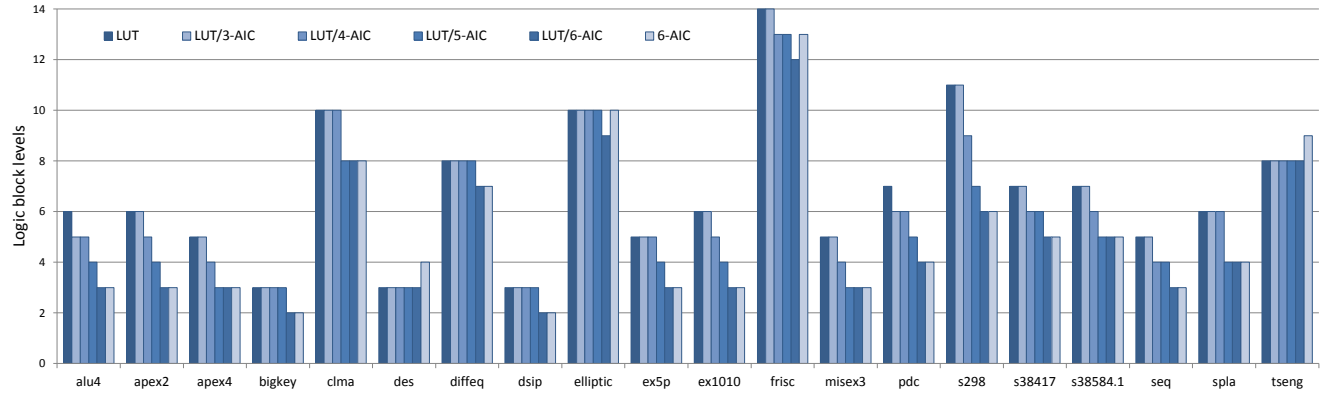


**Figure 8: Number of logic blocks (both LUTs and AICs) on the critical path.**

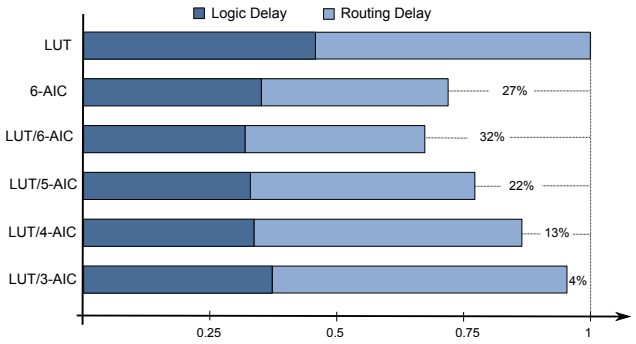


**Figure 9: Geometric mean of normalized total logic and routing delays.**

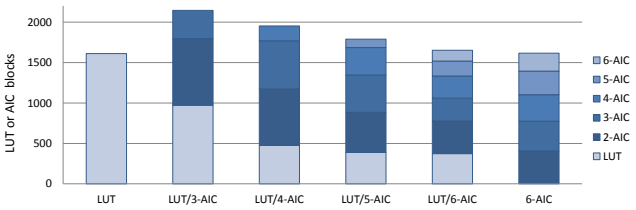


**Figure 10: Number and type of logic blocks used in the various architectures and with the various mapping strategies.**

total average logic and routing delays of the circuits. On average, the implementations on the pure 6-AIC architecture and on the hybrid architecture with 6-AIC and 5-AIC base blocks are 27%, 32%, and 22% faster than the baseline FPGA, respectively.

Fig. 10 presents the distribution of LUTs and AICs for the different architectures. This figure shows that when deeper cones are allowed, less LUTs are used. Moreover, in each case the usage of each AIC type has a reverse relation with the size of the AIC. This means that the chance of mapping a node with smaller AIC is always higher. Since each of these LUTs and AICs are packed into clusters, the numbers presented there do not indicate the real logic area of the circuits. On the contrary, Fig. 11 illustrates the number of clusters after packing: this is proportional to the active area since the area of an AIC cluster is close to the area of a LAB (see Table 2) and both have the same I/O bandwidth. For some benchmarks, either the *LUT/6-AIC* hybrid architec-

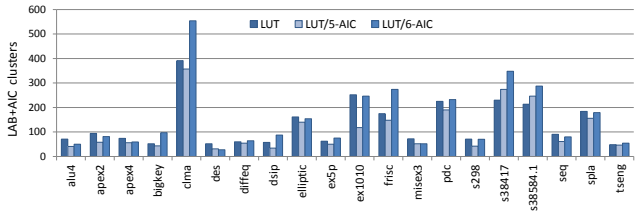


**Figure 11: Area measured as the total number of clusters used, completely or partially. LABs and AIC clusters occupy approximately the same area. On average, LUT/5-AIC uses 16% less resources than LUT-only.**

| Benchmark | LUT | LUT/ 5-AIC | LUT/ 6-AIC |
|---|---|---|---|
| alu4 | 14.9 | 10.59 | 11.32 |
| apex2 | 16.4 | 15.2 | 12.9 |
| apex4 | 15.5 | 16.1 | 14.1 |
| bigkey | 14.3 | 12.6 | 11.6 |
| clma | 20.8 | 22.9 | 25.5 |
| des | 14.6 | 16.1 | 15.1 |
| diffeq | 10.4 | 13.4 | 13.8 |
| dsip | 18.6 | 17.4 | 12.5 |
| elliptic | 15.5 | 16.6 | 16.7 |
| ex5p | 11.2 | 15.9 | 23.2 |
| ex1010 | 23.8 | 18.2 | 30.3 |
| frisc | 18.8 | 19.35 | 23.2 |
| misex3 | 14 | 12 | 13 |
| pdc | 22.8 | 23.4 | 21.2 |
| s298 | 13.2 | 9.7 | 15.8 |
| s38417 | 12.5 | 18.2 | 19 |
| s38584.1 | 11.5 | 18.4 | 17.5 |
| seq | 17.1 | 15.5 | 15.5 |
| spla | 21.5 | 18.8 | 21.1 |
| tseng | 8.3 | 13.1 | 12.5 |

**Table 5: Average wire length in units of one CLB segments.**

ture or the baseline FPGA display the lowest area; however, the *LUT/5-AIC* architecture always results in the smallest used area at a much better delay than the baseline FPGA and a slightly worse one than *LUT/6-AIC*—refer to Fig. 9. The two hybrid architectures define Pareto optimal points.

The hybrid structure of the proposed FPGA with the different cluster types needs to fix the right ratio of the two flavors of logic blocks. The packing results indicate that this ratio varies from one circuit to the other, making this problem not straightforward. We have made some preliminary experiments on this front, and we have fixed the ratio of LAB columns to AIC clusters to 1:4. The advantage of AICs is that any logic function that is mapped to a LUT is mappable to one or more AICs. The reverse is also true. Therefore, it is possible to switch to another logic block type, when we run out of one type. Moreover, considering the small size of the AIC blocks, it is quite feasible to add them as *shadow blocks* of the LUTs to the LUT clusters, by reusing the existing input crossbar. This provides the option to use either LUTs or AICs depending on the requirements. Though, adding AICs as shadow blocks of LUTs remains as the future work.

Table 5 presents the average wire length of each benchmark, in the baseline architecture (no AIC clusters) and in the two best hybrid architectures, with the number of routing channels fixed to 180 for all the experiments. We observe that there is a fairly high variability—but averages are very similar (15.8, 16.1, and 17.3 respectively)—with a small trend against our hybrid architecture.

# 8. RELATED WORK

Leveraging the properties of logic synthesis netlist to simplify the logic block of FPGAs is a current research topic [2, 3]. For instance, based on the observation that circuits represented using AIGs frequently have a trimming input, a low-cost and still LUT-based logic block was designed that requires less silicon area, but it does not improve the delay [3]. Albeit somehow similar in its inspiration to modern synthesis, our work is more radical in using the AIGs to inspire the new logic cell.

Although LUT-based logic blocks dominate the architectures of commercial FPGA, PAL-like logic blocks have also been explored. In recent times, it has been shown that a fairly small PAL-like structure, with 7–10 inputs and 10–13 product terms, obtains performance gains at the price of an increase in area [8]. Much earlier, some authors have shown that $K$-input multiple-output PAL-style logic blocks are more area efficient than 4-input LUTs. However, the idea was abandoned because PAL-based implementations typically consumed excessive static power [14]. Our solution moves away from the typical logic block natural of traditional logic synthesis, and we have shown that it seems possible to improve both area and delay compared to LUT-based FPGAs.

There are also numerous pieces of work which have adapted or created reconfigurable logic blocks to specific needs, often by adding dedicated logic gates to existing LUTs. Among these, one can mention GARP [10] and Chimaera [25] for datapath oriented processor acceleration, macro gates [12] for implementing wide logic gates, and various sorts of fast carry chains beyond those available commercially [21]. Although they all somehow question the pure LUT as the most efficient building block, they tend to introduce modifications that are never real generic alternatives.

# 9. CONCLUSIONS

As several people before us, we have recognized that LUTs have many advantages but, frequently, the price to pay for these advantages is unreasonably high. We have thus explored new logic blocks inspired by recent trends in the circuits representations used in logic synthesis: we came to define AICs which are simply the natural configurable circuits homologue of the newly popular AIGs. We have explored alternate FPGAs architectures based on these AICs, essentially fitting the new logic block into a traditional FPGA architecture without changing some global parameters whose impact would be very difficult for us to master. Despite these artificial limitations, we find first results encouraging: On one hand, delay is bound to decrease as both logic delay and the number of logic blocks on the critical path reduce. With a fairly rough routing delay model we observe a delay reduction of up to 32%. On the other hand, the number of

logic blocks (all of similar area) consumed by the benchmark circuits is also generally reduced; with one of our mapping approaches, the area is reduced on average by 16%. Future work will necessarily need to address placement and routing much more precisely than we had the chance to. Also, other less conservative architectures may prove more advantageous than those explored. Nevertheless, we think that our first results are sufficiently encouraging for the approach to deserve a closer inspection.

# 10. REFERENCES


[1] Altera Corporation. *Stratix II Device Handbook, vols. 1 and 2*. `http://www.altera.com/literature/`.

[2] J. H. Anderson and Q. Wang. Improving logic density through synthesis-inspired architecture. In *Proceedings of the 19th International Conference on Field-Programmable Logic and Applications*, pages 105–11, Prague, Aug. 2009.

[3] J. H. Anderson and Q. Wang. Area-efficient FPGA logic elements: Architecture and synthesis. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 369–75, Yokohama, Japan, Jan. 2011.

[4] Berkeley Logic Synthesis and Verification Group, Berkeley, Calif. *ABC: A System for Sequential Synthesis and Verification*, Feb. 2011. Release 10216, `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[5] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic, Boston, Mass., 1999.

[6] J. Cong and Y. Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. In *Proceedings of the International Conference on Computer Aided Design*, pages 49–53, Santa Clara, Calif., Nov. 1992.

[7] J. Cong and Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(2):137–48, June 1994.

[8] J. Cong and H. Huang. Technology mapping and architecture evaluation for k/m- macrocell-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(1):3–23, Jan. 2005.

[9] J. Cong, C. Wu, and Y. Ding. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the 7th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 29–35, Monterey, Calif., Feb. 1999.

[10] J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *Proceedings of the 5th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 12–21, Napa Valley, Calif., Apr. 1997.

[11] L. Hellerman. A catalog of three-variable Or-Invert and And-Invert logical circuits. *IEEE Transactions on Electronic Computers*, EC-12(3):198–223, June 1963.

[12] Y. Hu, S. Das, S. Trimberger, and L. He. Design, synthesis, and evaluation of heterogeneous FPGA with mixed LUTs and macro-gates. In *Proceedings of the International Conference on Computer Aided Design*, pages 188–93, San Jose, Calif., Nov. 2007.

[13] A. Kaviani and S. D. Brown. Hybrid FPGA architecture. In *Proceedings of the 4th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 3–9, Monterey, Calif., Feb. 1996.

[14] J. L. Kouloheris and A. El Gamal. PLA-based FPGA area versus cell granularity. In *Proceedings of the IEEE Custom Integrated Circuit Conference*, pages 4.3.1–4.3.4, Boston, Mass., May 1992.

[15] Y. Kukimoto, R. Brayton, and P. Sawkary. Delay-optimal technology mapping by DAG covering. In *Proceedings of the 35th Design Automation Conference*, pages 348–51, San Francisco, Calif., June 1998.

[16] I. Levin and R. Y. Pinter. Realizing expression graphs using table-lookup FPGAs. In *Proceedings of the 30th Design Automation Conference*, pages 306–11, Dallas, Tex., June 1993.

[17] D. Lewis et al. The Stratix II logic and routing architecture. In *Proceedings of the 13th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 14–20, Monterey, Calif., Feb. 2005.

[18] J. Luu, J. H. Anderson, and J. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 227–36, Monterey, Calif., Feb. 2011.

[19] V. Manohararajah and S. Brown. Heuristics for area minimization in LUT-based FPGA technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2331–40, Nov. 2006.

[20] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Design Automation Conference*, pages 532–36, San Francisco, Calif., July 2006.

[21] H. Parandeh-Afshar, P. Brisk, and P. Ienne. An FPGA logic cell and carry chain configurable as a 6:2 or 7:2 compressor. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2(3):19:1–19:42, Sept. 2009.

[22] M. Schlag, J. Kong, and P. K. Chan. Routability-driven technology mapping for lookup table-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):13–26, Jan. 1994.

[23] H. Yang and D. F. Wong. Edge-map: Optimal performance driven technology mapping for iterative LUT based FPGA designs. In *Proceedings of the International Conference on Computer Aided Design*, pages 150–55, San Jose, Calif., Nov. 1994.

[24] S. Yang. Logic synthesis and optimization benchmarks user guide, version 3.0. Technical report, Microelectronics Center of North Carolina, Research Triangle Park, N.C., Jan. 1991.

[25] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. CHIMAERA: A high-performance architecture with a tightly-coupled reconfigurable functional unit. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 225–35, Vancouver, June 2000.