The Hybrid Field-Programmable Architecture

ALIREZA KAVIANI
STEPHEN BROWN
University of Toronto

The authors propose a new architecture that combines two existing technologies: lookup-table-based FPGAs and complex programmable logic devices based on PLA-like blocks. Their mapping results indicate that on average LUT-based FPGAs require 78% more area than their hybrid FPGA, while providing roughly the same circuit depth.

FOR THE PAST SEVERAL YEARS, high-capacity field-programmable devices have enjoyed a rapidly expanding market and have become widely accepted for the implementation of small to moderately large digital circuits. The two main types of FPDs, field-programmable gate arrays and complex programmable logic devices, are both widely used, each offering particular strengths. FPGAs programmed with static RAM technology are usually based on lookup tables. Their main strengths are very high logic capacity—in the range of hundreds of thousands of equivalent logic gates—and good speed-performance—up to 50-MHz system clock rates. On the other hand, CPLDs consist of multiple PLA-like blocks, in which the OR planes are partly fixed. Their characteristics include medium capacity, in the range of a few thousand gates, and ultrahigh speed-performance, sometimes in excess of a 200-MHz system clock rate.

In this article, we propose a new FPD architecture, called the Hybrid Field-Programmable Architecture, which combines FPGAs and CPLDs. The basis of the HFPA is that some parts of digital circuits are well-suited for implementation with LUTs, but other parts benefit more from the product-term-based structures in CPLDs. Comparison with an architecture containing only LUTs indicates that the new architecture offers significant savings in total chip area. Also, the HFPA can reduce the depth of circuits implemented in the FPGA, which may provide improvements in speed-performance.

Underlying benchmark analysis

We can represent any digital circuit as a directed acyclic graph consisting of combinational and sequential nodes, with each combinational node of the circuit in sum-of-products form. As the first step in developing the HFPA, we examined the combinational nodes present in real circuits and produced a distribution of the nodes with respect to size. We defined a node's size according to two parameters: the number of inputs to the node and the number of product terms in the node's sum-of-products representation. The circuits we used are from the 1993 MCNC (Microelectronics Center of North Carolina) logic synthesis benchmark suite. We passed the circuits through one run of SIS script.rugged,[1] a technology-independent optimization script. This resulted in 40,131 combinational nodes in 197 benchmarks. Our examination of the nodes revealed that more than 70% are 4-bounded and roughly 20% have fan-ins equal to or greater than six; we refer to the latter as high-fan-in nodes.

We wanted to consider implementing the nodes in two types of logic resources: PLA-like blocks and LUTs. For a LUT with K in-

0740-7475/99/$10.00 © 1999 IEEE

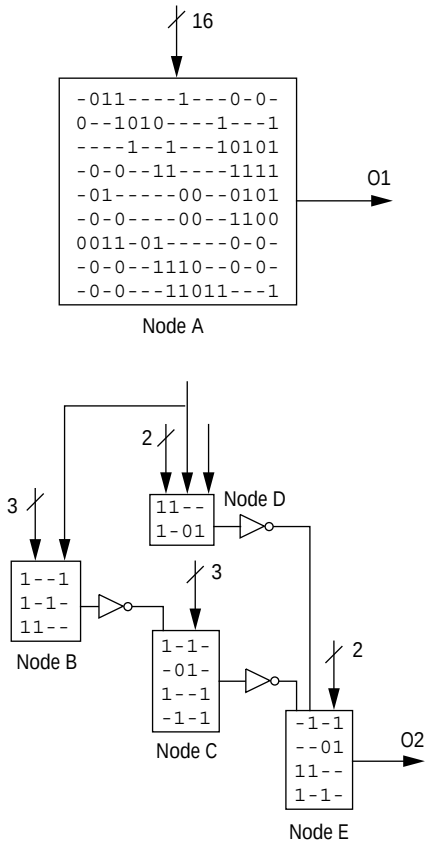

Figure 1. *An example of combinational nodes in a circuit.*

puts, the cell's area is proportional to 2^K. For a PLA-like block, the area is approximately proportional to $K \times P$, where P is the number of pterms in the block. Assuming P is close to K, we can simplify the representative area of a PLA-like block to K^2. For $K = 4$, $2^K = K^2$, and $K < 4$, LUTs are more efficient than PLAs. Therefore, 4-bounded nodes can be efficiently implemented with LUTs. This accounts for most nodes in circuits, but there is still a significant number with high fan-ins. We could also implement these nodes with 4-LUTs, but the area required would be large. We observed that most high-fan-in nodes do not require a large number of pterms and thus are well suited for PLA implementation. Therefore, we decided the HFPA would contain both PLA-like blocks, which we call programmable array logic blocks, and 4-LUTs.

Figure 1 illustrates the concept of suitability of nodes of different sizes in either LUTs or PLAs. The example circuit consists of five combinational nodes, each represented by its personality matrix. The personality matrix's columns are associated with the inputs to the combinational node, and its rows correspond to the pterms in the sum-of-products form of the node. For example, a 1 in the second row and fourth column means that the fourth input to the node appears in the second pterm of its sum-of-products form, with positive polarity. Similarly, a 0 at this position means that the corresponding literal appears with negative polarity, and the symbol "–" means that the corresponding literal does not appear in the pterm. Therefore, the numbers of rows and columns in a personality matrix equal the numbers of pterms and inputs in the sum-of-products form of the node, respectively. Note that we are not considering the inverters as separate nodes because they can be realized in their fan-in cells with no extra cost.

Abbreviations

BLIF	Berkeley Logic Interchange Format
CAM	content-addressable memory
CPLD	complex programmable logic device
EDIF	Electronic Design Interchange Format
FPD	field-programmable device
FPGA	field-programmable gate array
HFPA	the Hybrid Field-Programmable Architecture
LUT	lookup table
4-LUT	four-input LUT
LUTB	LUT block, contains four 4-LUTs that can be locally interconnected
PALB	programmable array logic block (a PLA-like block)
PLA	programmable logic array
pterm	product term

If we implement the circuit shown in Figure 1 in a LUT-based FPGA, we will need four 4-LUTs for nodes B, C, D, and E in addition to the number of LUTs required to implement node A. We used the Synopsys FPGA compiler with the highest optimization effort to map node A to 4-LUTs; we needed a total of 15 LUTs. Therefore, we needed at least 19 LUTs to implement the five combinational nodes. In an architecture that contains PALBs as well as LUTs, we can implement node A in one PALB, and the rest of the nodes will require four 4-LUTs. This is the equivalent area of only eight 4-LUTs, as explained later. The key point is that when synthesis tools fail to find good decompositions for a node such as A, the area necessary to realize the node with LUTs is high.

On the other hand, O2 in Figure 1 is efficiently decomposed into four 4-bounded nodes; therefore 4-LUTs can implement O2 in a reasonable area. If we fully collapse nodes B to E, the result will have 12 inputs and 26 pterms. Such a large number of pterms is expensive in terms of area when implemented in a PLA-like block. In a CPLD containing only PLA-like blocks of the size of our PALBs, the circuit in Figure 1 will take at least 2.5 PALBs, equivalent to 10 4-LUTs in area. Therefore, an architecture that contains a mixture of PALBs

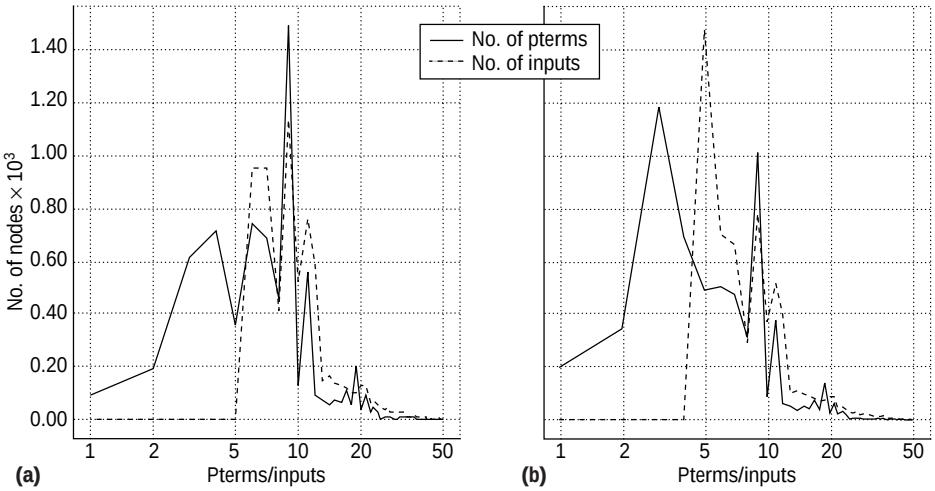

Figure 2. *Node size distribution: excluding 5-bounded nodes* ***(a)****; excluding 4-bounded nodes* ***(b)****.*

Table 1. *Architectural statistics. μ = mean; σ = variance; boldface numbers = best choice.*

Excluding *K*-bounded nodes	Inputs (μ, σ)	Pterms (μ, σ)	Inputs / pterms (μ, σ)	Pterms (filtered) (μ, σ)	Inputs / pterms (filtered) (μ, σ)
K = 6	(12.8, 6.8)	(9.2, 5.3)	(1.6, 0.9)	(6.3, 5.5)	(1.9, 1.0)
K = 5	(11.8, 6.7)	(8.4, 5.3)	(1.6, 0.9)	(5.8, 5.2)	(1.9, 1.0)
K = 4	(10.2, 6.6)	(7.2, 5.2)	(1.7, 0.9)	**(5.0, 4.8)**	**(1.9, 0.9)**
K = 3	(8.2, 6.1)	(5.5, 4.9)	(1.8, 0.9)	(4.1, 4.2)	(1.9, 0.7)
K = 2	(6.2, 5.4)	(4.1, 4.3)	(1.8, 0.9)	(3.1, 3.5)	(1.8, 0.6)
K = 0	(4.5, 4.7)	(3.0, 3.6)	(1.7, 0.7)	(2.3, 2.9)	(1.7, 0.6)

and LUTs provides the most area-efficient implementation for this example.

Design process

Previous research[2] has studied the effects of LUT size on FPGA area efficiency and concluded that 4-LUTs provide good results. Here, we assume that the HFPA uses 4-LUTs. This section explains the analysis that led to our determination of the number of inputs, pterms, and outputs for the PALB.

Analysis of node sizes in benchmark circuits. Since 4-bounded nodes will be implemented in 4-LUTs, the PALB should be designed in such a way that it is well matched for implementation of nodes with more than 4 inputs. Also, we observed that many 5-input nodes are simple OR/AND gates; these nodes can easily be decomposed and realized in 4-LUTs. It is not desirable that the nodes to be implemented in LUTs affect the PALB architecture.

Figure 2a shows the node size distribution, in terms of number of pterms and number of inputs, for all the MCNC benchmarks, excluding 5-bounded nodes. The figure shows a peak at 9 pterms, with some larger nodes and many smaller ones. To illustrate the effects of 5-input nodes, Figure 2b shows the same information, except that only 4-bounded nodes are excluded. Now, there is a new peak at 3 pterms; it is clear that the number of 5-input nodes is significant and may strongly affect our analysis. Because the effect of 5-input nodes is pronounced, and because many of these nodes are suitable for LUTs, it is important to exclude them when designing the PALB architecture. The same is true for nodes with a higher fan-in, but their effect would be less important because there are fewer of them.

Table 1 summarizes statistics of node sizes excluding nodes with various numbers of inputs. Three statistical parameters affect the PALB: 1) the average number of inputs, 2) the average number of pterms, and 3) the ratio of the number of inputs to the number of pterms for each combinational node. Table 1 gives the mean and variance (μ, σ) of these parameters. Each row corresponds to a specific value of *K*, showing the statistical data excluding *K*-bounded nodes. Under the columns labeled "filtered," additional nodes are excluded according to the following assumptions: 1) All single-input pterms in a node merge into one multi-input pterm. We base this assumption on our observation that in the sum-of-products form of high fan-in nodes, there are many pterms with only one input. As explained in the next subsection, we can merge these single-input pterms into one, with almost no extra cost for the PALB architecture. 2) Nodes that are single pterms (that is, ANDs and ORs) are excluded. There are many combinational nodes with only one pterm. Since these nodes are decomposable, LUTs are as good as PALBs for implementing them, so we must exclude them from the data that affect the PALB architecture.

Table 1 serves as a guide for designing the PALB archi-

tecture. Since we are using 4-LUTs, it is reasonable to base the PALB on the $K = 4$ row, and for reasons discussed earlier, it is appropriate to use the filtered columns. Thus, the PALB should be designed to suit the parameters shown in boldface. We decided that 5 and 1.6 are the closest practical values to the boldface averages.

PALB architecture. Figure 3 shows the PALB we developed for use in the HFPA. It has 16 inputs, 10 pterms, and 3 outputs. On average, two combinational nodes, each with 5 pterms, can be implemented in a PALB. One extra output accommodates the implementation of small nodes. We discuss the implementation of AND/OR gates later.

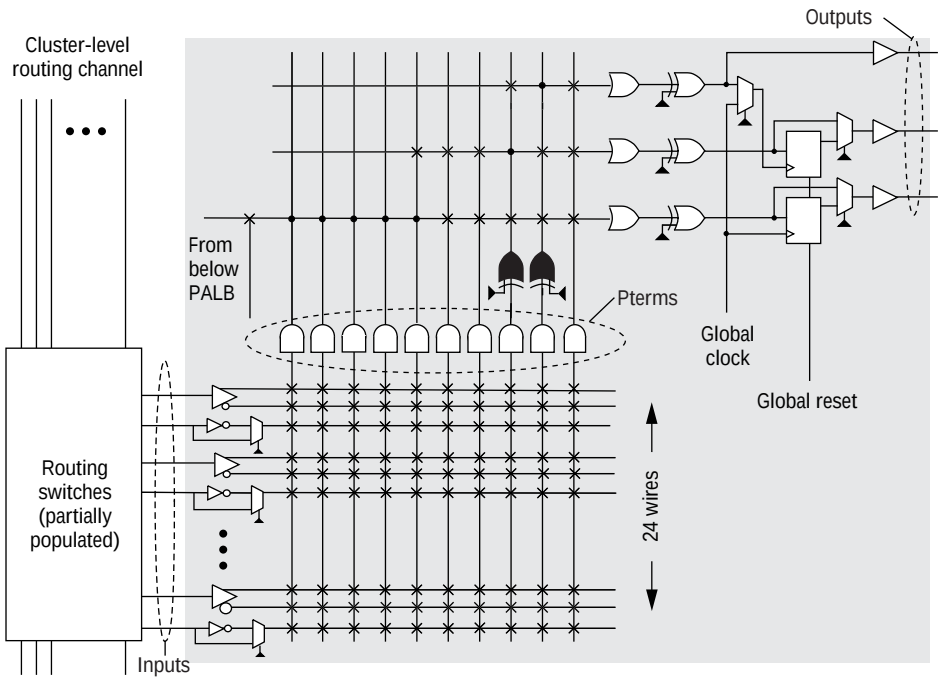

Figure 3. *PALB architecture.*

The meaning of the schematic in Figure 3 should be readily apparent, except for the connection of the inputs to the OR plane. The figure shows 5 pterms hardwired to produce the lowest of the 3 outputs (marked •). Also, another 5 pterms can be programmably connected to this output (marked x). Similar comments apply to the other PALB outputs. This architecture combines a classic fixed OR plane with a programmable OR plane. It is designed to be optimized for 5 pterms and yet be configurable for up to 10 pterms. One of the outputs in the PALB accommodates an extra programmable input to the corresponding OR. We can connect this input to the adjacent PALB, a feature useful for implementing nodes with more than 10 pterms. Figure 3 illustrates that for half of the PALB's inputs, we can connect only one polarity to the AND plane. The analysis of the personality matrices of the combinational nodes in the benchmarks supports this. According to our analysis, less than 1% of PALB-feasible combinational nodes need more than 8 inputs with both polarities. These few nodes can be implemented in 4-LUTs. By using this feature, we reduce the number of input switches by 25%, significantly reducing the PALB's silicon area.

Figure 4 depicts two functionally equivalent implementation forms of a typical combinational node. Figure 4b shows the merging of pterms with single inputs into one multi-input pterm. Note that the extra AND gate and inverters at its inputs are already available in our PALB, at no additional cost. In this example, the number of pterms decreases from 5 to 3 with the cost of one extra inverter. The black XOR gates in Figure 3 serve as programmable inverters for the purpose of merging single-input pterms into one multivariable pterm. The data in the "filtered" columns in Table 1 justify this PALB feature.

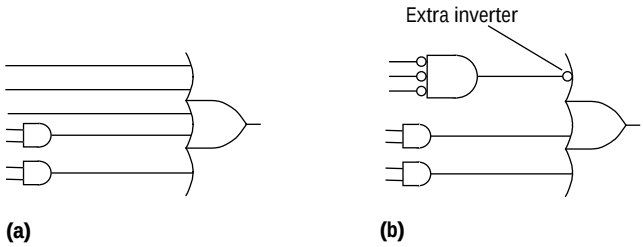

Figure 4. *Node implementation forms: AND-OR* ***(a)****; AND-OR plus merged single-input pterms* ***(b)****.*

The simple act of merging decreases the number of pterms in the observed 197 benchmarks approximately 25% on average. This is clear in Figure 5 (next page), which shows the average number of pterms with and without merging. The merging feature thus reduces the logic resources needed to implement circuits, contributing to better area efficiency. For completeness, we should mention that the benefits of pterm-merging depend on the particular logic optimization methods used to produce the set of nodes in

the circuits.

Routing architecture and layout issues

The choice of an appropriate routing architecture is extremely important in designing an FPD. A custom PALB layout determines the block's area and delay.

Interconnection structure. The HFPA's routing channels follow a hierarchical arrangement. A previous study[3] has shown that such routing architectures are efficient. Commercial FPGAs such as Altera's Flex 10K use hierarchical routing schemes.

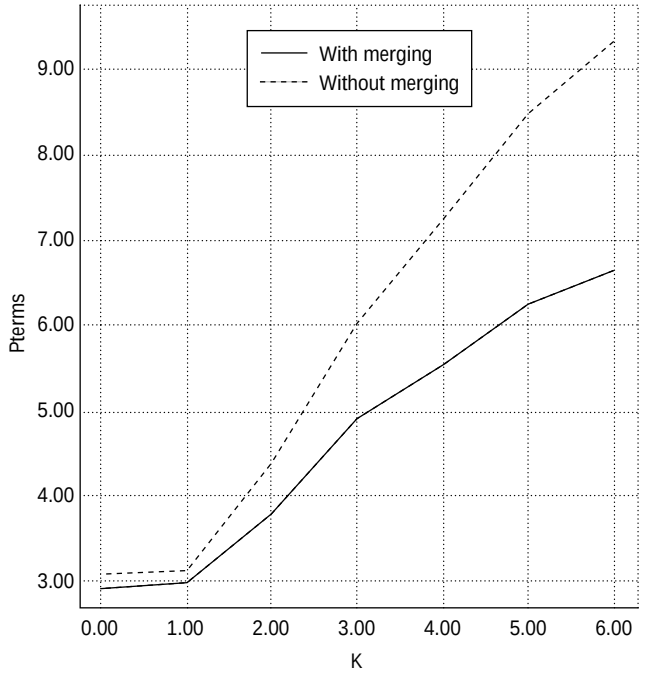

Figure 5. *Average number of pterms.*

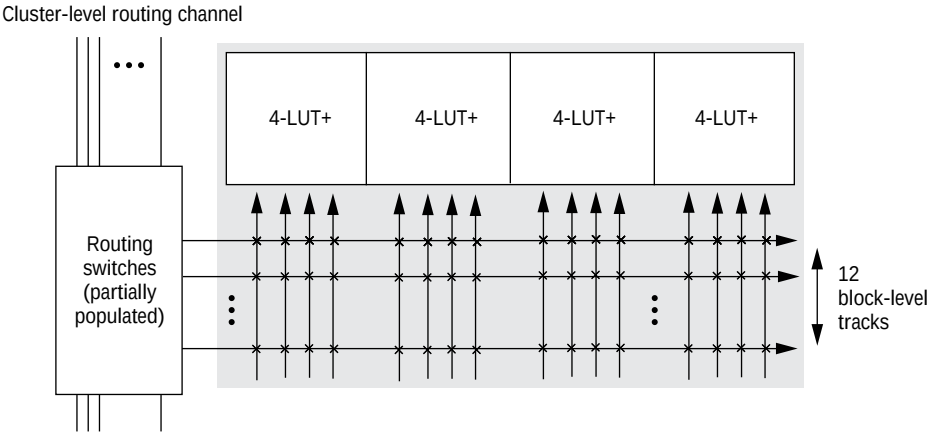

Figure 6. *A LUTB in the HFPA.*

The lowest level in the HFPA is a block, which can be a PALB or a LUTB. A LUTB contains four 4-LUTs that can be locally interconnected, as illustrated in Figure 6. Each 4-LUT+ cell in the figure can accommodate the equivalent functionality of a 4-input LUT and also contains a flip-flop and necessary drivers and switches for the cell's output. Since a 4-LUT+ is similar to cells used in traditional FPGAs, we will not explain its details. Figure 6 shows that each cell is connected to a block-level channel through a fully populated crossbar. We implement these crossbars with multiplexers; therefore, it is feasible in terms of area to fully populate them. Higher flexibility in lower levels of routing allows us to reduce the number of switches in upper levels, whose transistors drive larger loads and therefore require more area. The block-level routing channel contains 12 tracks. A partially populated crossbar connects the 12 tracks to a cluster-level channel.

Figure 7 shows a cluster containing eight PALBs and eight LUTBs. The number of blocks and the mixture of PALBs and LUTBs in the figure are tentative; future research must confirm these parameters. There are 80 tracks in the cluster-level routing channel, and each track can connect to two wires of each block. Thus, each partially populated cluster-to-block crossbar has 160 switches. We decided that two switches per track in the cluster channel are sufficient to provide reasonable routability, because all the inputs to a block are functionally equivalent. This is true for both LUTBs and PALBs because the block-level crossbars are fully populated. The positioning of switches shown in Figure 7 simplifies illustration but is not what we used in the layout.

We can connect several clusters to create a section using another level of routing. We can homogeneously distribute the switches and section-level routing tracks among the PALBs and LUTBs in a cluster. Therefore, the area needed for higher levels of routing is the same for LUTBs and PALBs. Our main objective is to come up with realistic area and delay estimates within a section. Thus, we do not need to discuss the details of routing characteristics and resource layout for levels higher than the cluster-level channel.

PALB layout. To determine a PALB's area and delay, we made a custom layout of the block in a 0.5-micron CMOS process that allows three levels of metallized interconnect. Instead of manual layout of the block, we used Ballistic,[4] an analog/digital layout language developed at the

University of Toronto. The Ballistic compiler is mounted on top of the Mentor Graphics GDT layout editor, which produces the final layout. We do not explain the details of the layout here, but we describe the main issues involved in the block's area and delay.

Figure 8 shows our implementation of AND/OR gates in the PALB. Instead of the more common pseudo-NMOS AND gates used in commercial CPLDs, we opted for static CMOS gates. This allows us to put many PALBs on one chip without concern for static power consumption. To reduce the delay of the AND/OR gates and thus the block's total delay, we implemented the gates in two levels. The total area of the AND/OR gates is less than 7% of the whole PALB, a negligible area penalty.

Our goal was to reduce the PALB's area and delay as much as possible. To minimize the area, we needed to obtain a perfect balance between the diffusion layer and all the metal interconnection levels. According to our observations of previous submicron FPD layouts, the area bottleneck is often one of the metal layers. Therefore, we focused on reducing the interconnections that needed metal-2 and metal-3 levels by placing the elements so that most connections were locally feasible using polysilicon and metal-1 layers.

With the same intentions in mind, we decided to use five-transistor SRAMs for our programming bits. Figure 9a illustrates a typical SRAM programming bit for the AND input. We used two lines to program the SRAM: pgmctl controls the programming and pgmin provides the programming data for each SRAM cell. The programming cell produces a 1 for the AND input when the pass transistor is off. A similar programming cell generates a 0 for the OR input when the pass transistor is off.

We did not implement the inputs that provide both polarities to the AND plane in the manner shown in Figure 3. Instead, we used one programmable inverter for each AND input with both polarities. This reduces the number of level-2 wires with a negligible increase in the area needed in the silicon level. Figure 9b shows a programmable inverter. The AND gates in the layout are surrounded by both kinds of programming cells shown in Figure 9.

Figure 10 (next page) presents the layout's dimensions and overall placement of elements. The numbers are in microns, truncated to the closest integer for simplicity. Cluster-block routing takes roughly 30% of the PALB's area. SRAM

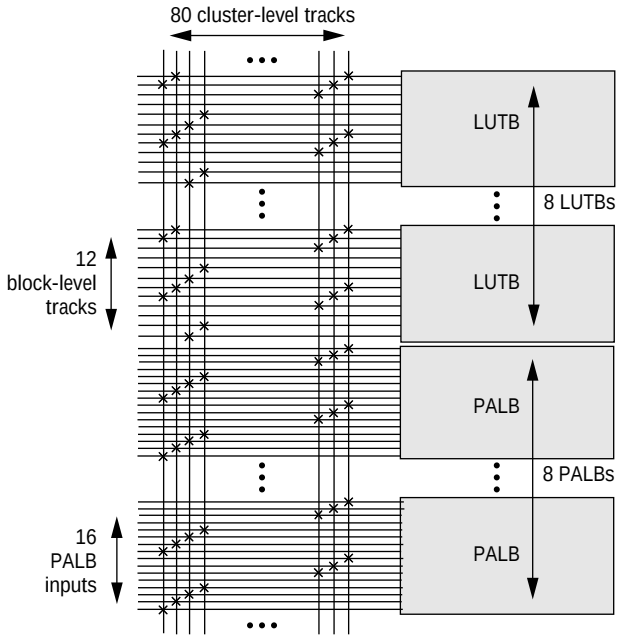

Figure 7. *A cluster in the HFPA.*

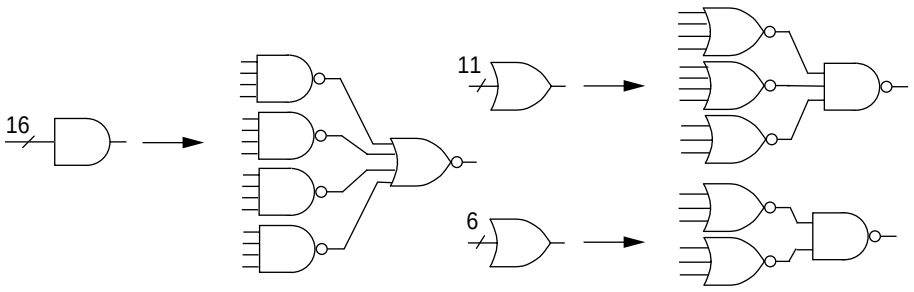

Figure 8. *PALB implementation of AND/OR gates.*

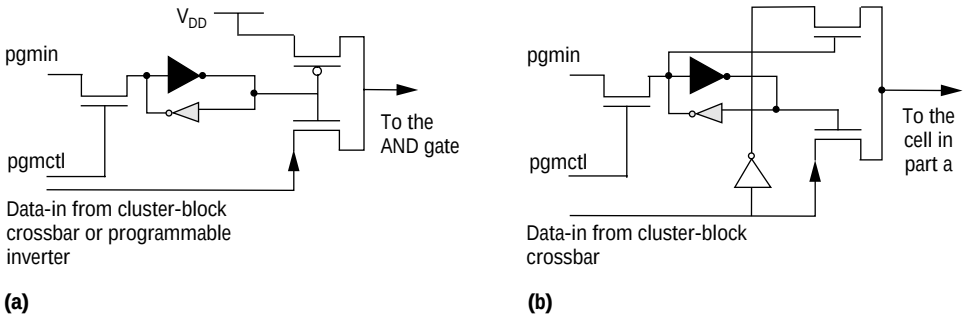

Figure 9. *Programming cells: SRAM cell* ***(a)****; programmable inverter* ***(b)****.*

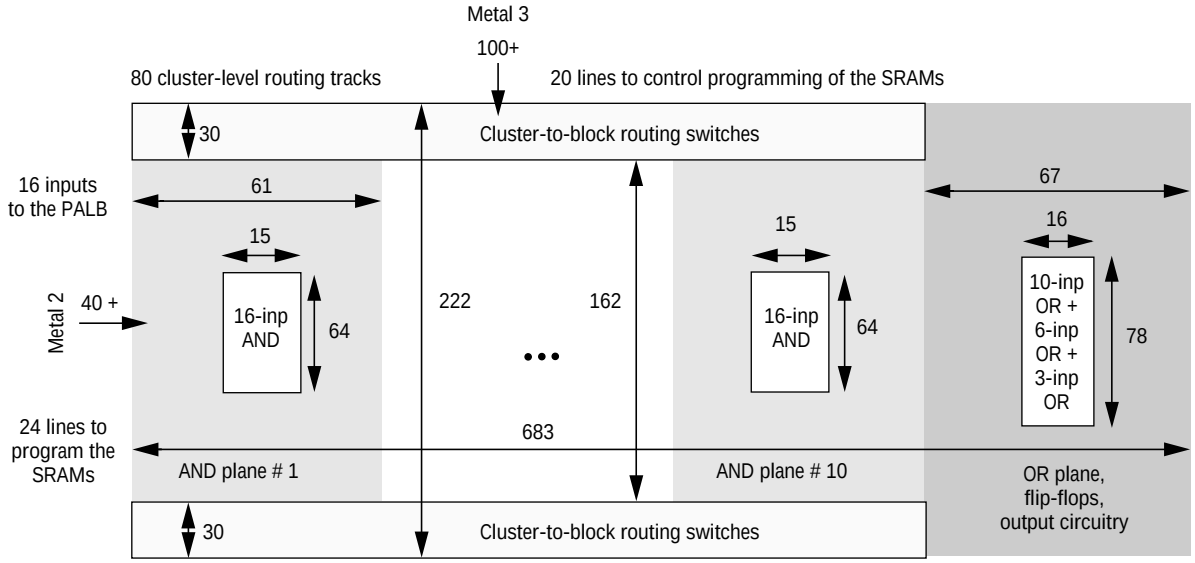

Figure 10. *PALB layout. Dimensions are in microns.*

programming cells in the input AND planes use more than 50% of the area. The AND gates use roughly 6% of the total area, showing that implementing them with static CMOS gates (Figure 8) is a good approach. The PALB's total area is slightly more than 0.15 mm^2.

The LUTB layout in the same technology process occupies an area of about 0.18 mm^2. We also estimated the area of a 4-LUT in the same process, using information provided by industry. Our comparison found that the PALB layout's area is somewhat smaller than that of the LUTB layout from industry. In the next section, we assume that both blocks take the same area; this is a pessimistic assumption for the HFPA. Also, our postlayout simulation results for 3.3 V_{DD} indicate that the PALB's delay from cluster tracks to the outputs before drivers is less than 2.5 ns. This is very close to the 4-LUT's delay in commercial LUT-based FPGAs. For example, in the Altera EPF6016, the maximum delay of a logic element that fans out to another logic element in the same logic array block is 3.2 ns. The EPF6016 is also built in a 0.5-micron process. Measuring speed-performance is a complicated task because the delay between two LUTs in the same LUTB differs from the delay between them in a cluster or section. In the next section, we assume that a PALB has the same delay as a LUTB, and we measure the depth of the circuits. This simplification gives us an estimate of the HFPA's speed-performance improvement.

Estimate of gain

An accurate measure of the gain provided by the new architecture requires CAD tools for technology mapping, placement, and routing of the circuits. These tools are not yet available, so we approximated the HFPA's expected gain compared to a 4-LUT-based FPGA. Table 2 lists the results of the comparison for 11 MCNC benchmarks. We converted the benchmarks from EDIF to BLIF,[1] and after optimization, mapped them to 4-LUTs using the Flowmap mapping algorithm.[5] All the benchmarks passed through one run of the SIS technology-independent optimization scripts. We applied two optimization scripts: script.rugged and script.algebraic. Both scripts use various methods of optimizing for combinational circuits. The HFPA's gain was lower when we mapped the circuits after using script.algebraic, so to be conservative, we report in Table 2 only the script.algebraic results. We discuss the effects of technology-independent optimization on the HFPA later.

The "4-LUT" column of the table shows the number of 4-LUTs and the depth of the circuit after technology mapping using Flowmap. We define the depth of a circuit as the total number of LUTs and PALBs on the circuit's critical path. To estimate the HFPA's relative gain after technology-independent optimization, we partially collapsed the benchmarks, using the SIS eliminate and reduce_depth commands,[6] and then mapped them to the new architecture, using HFAmap.[7] We used the eliminate command to minimize area, and we applied reduce_depth when depth was our concern. The SIS partial collapsers are not ideal for the HFPA, but they provided acceptable results, as Table 2 shows.

Two area-depth trade-offs illustrate the HFPA's advantages. The table's "HFPA-area" columns list the number of PALBs and 4-LUTs resulting from mapping the circuits to the HFPA such that the mapped circuit's depth is close to that obtained with Flowmap. The results in the "HFPA-

Table 2. *Estimate of HFPA gain. HFPA-area: optimized for area; HFPA-depth: optimized for depth; depth: number of LUTs and PALBs on critical path; area: equivalent 4-LUTs*

MCNC benchmark	No. of 4-LUTs (area, depth)	HFPA-area Area	HFPA-area Depth	HFPA-area Gain % (area, depth)	HFPA-depth Area	HFPA-depth Depth	HFPA-depth Gain % (area, depth)
s1423	221, 15	$20 \times 4 + 63 = 143$	14	55, 7	$41 \times 4 + 33 = 197$	8	12, 88
s1488	296, 4	$25 \times 4 + 50 = 150$	4	97, 0	$27 \times 4 + 49 = 157$	3	89, 33
frg2	370, 6	$24 \times 4 + 103 = 199$	5	86, 20	$54 \times 4 + 81 = 297$	4	25, 50
x3	377, 5	$33 \times 4 + 82 = 214$	4	76, 25	$70 \times 4 + 38 = 318$	3	19, 67
dalu	500, 6	$41 \times 4 + 77 = 241$	5	107, 20	$39 \times 4 + 74 = 230$	4	117, 50
scf	410, 5	$33 \times 4 + 92 = 224$	6	83, –17	$42 \times 4 + 64 = 232$	3	77, 67
cordic	466, 9	$30 \times 4 + 170 = 290$	8	61, 13	$42 \times 4 + 86 = 254$	7	83, 29
sbc	372, 5	$37 \times 4 + 98 = 246$	6	51, –17	$49 \times 4 + 87 = 283$	3	31, 67
cps	749, 5	$55 \times 4 + 214 = 434$	5	73, 0	$84 \times 4 + 113 = 449$	3	67, 67
apex2	1878, 8	$176 \times 4 + 499 = 1,203$	8	56, 0	$253 \times 4 + 233 = 1,245$	6	51, 33
alu4	1522, 7	$135 \times 4 + 191 = 731$	8	108, –13	$171 \times 4 + 251 = 935$	6	63, 17
Average				78, 3.5			57, 52

depth" columns illustrate another trade-off, emphasizing the HFPA's advantage in reducing depth. The columns labeled "Area" show estimates of the HFPA's total chip area in terms of equivalent 4-LUT count, assuming each PALB takes an area equal to four 4-LUTs. This assumption is supported by the layout we described earlier. We calculated the area and depth gains as the percentage of increase in area/depth of the 4-LUT-based architecture compared to the HFPA's area/depth for the same circuit. We obtain the overall gain by taking the average of the gains of individual benchmarks.

The results in Table 2 imply various mixtures of LUTs and PALBs. We assume that enough resources of each type would be available in a real chip. The mixture is a function of the circuit's type and size. However, we believe that this issue is not critical for two reasons: First, LUTs and PALBs are interchangeable resources; LUTs can implement high-fan-in nodes if there are not enough PALBs, and PALBs can implement small nodes if too few LUTs are available. Second, a commercial HFPA product line would comprise various chips with different numbers of PALBs and LUTs. Our technology mapper accepts a balance factor that determines how to make the best use of available resources. It is easier to create the desirable balance for larger circuits. Kaviani[7] provides more details.

According to Table 2, LUT-based FPGAs are 78% less area-efficient than the HFPA, while offering roughly the same delay. The table also shows that one can map circuits to the HFPA such that the circuits mapped to 4-LUTs have a 52% higher depth on average. At the same time, they are still 57% less area-efficient than those mapped to the HFPA. Reducing the circuits' depth decreases not only the total delay due to blocks on the critical path, but also the number of switches needed to route those blocks. Reducing the number of switches, in turn, adds to the speed-performance improvement. For mapping to 4-LUTs, the comparisons given here are based on Flowmap because of its convenient availability as part of SIS and because it produces optimal results in terms of depth. Other algorithms[8] may produce a lower number of 4-LUTs with an increase in depth. This would reduce the area gain shown in the table, but we could expect a higher improvement in depth.

Unresolved issues and future work

Although the HFPA promises many advantages over traditional FPD architectures, several important issues still must be addressed.

Synthesis. For designers to use an FPD with a new architecture, the device should be accompanied by automated CAD tools. If synthesizing circuits in the new architecture is complicated, the industry might find it commercially unattractive. Although synthesis issues are beyond this article's scope, let us summarize how the HFPA affects the current FPGA synthesis flow. A typical synthesis flow technology maps a circuit after optimization. Then the circuit is partitioned into portions of desirable sizes. Finally, the partitioned circuit undergoes placement and routing.

Clearly, the HFPA complicates technology mapping. Our technology mapper HFAmap is discussed elsewhere[7] in more detail. Routing or partitioning tools can treat a PALB in the same way as a block containing several LUTs. For that

Related work

FPDs suffer from lower speed-performance and less logic capacity than custom-manufactured devices such as mask-programmed gate arrays. Therefore, much recent research has been devoted to improving FPD architecture. New applications continue to emerge as research in industry and academia results in more sophisticated products with higher total logic capacity and better speed-performance.

An early FPGA research study by Rose et al.[1] focuses on logic-block complexity. Assuming a LUT-based architecture, the authors varied the number of inputs to a LUT to measure the effects on implementation of a benchmark circuit set. Their conclusion is that LUTs with 4 or 5 inputs yield the best results in terms of chip area. We applied this result to the HFPA by using 4-LUTs, which are also found in commercial FPGAs such as the Altera Flex 10K, and the Xilinx Virtex.

Most FPD research has focused on FPGAs, and little work on CPLDs has appeared. However, Kouloheris and El Gamal[2] investigated FPDs built with PLA-based logic blocks. According to the authors, an FPD using PLAs with 10 inputs, 12 pterms, and 3 outputs achieves about the same level of logic density as FPGAs based on 4-LUTs. However, we are not aware of any commercial product using such PLAs.

A study by He and Rose[3] investigated heterogeneous FPGA architectures, which contain logic blocks of two different sizes. The researchers reported the effects on area efficiency of LUT-based FPGAs with two sizes of LUTs in the same chip. They found that a mixture of LUTs provides an average area savings of about 10%. Our HFPA is related to the heterogeneous FPGA in the sense that two different logic blocks are available. However, the heterogeneous FPGA has two sizes of the same type of logic block (LUTs), while the HFPA has two entirely different types of logic resources (LUTs and PLA-like blocks).

Wilton, Rose, and Vranesic[4] proposed including memory modules with variable aspect ratios as separate blocks in an FPGA. Tau et al.[5] proposed using multicontext programming bits, a scheme that promises some savings in area efficiency and reconfiguration time for FPGAs. These ideas are not orthogonal to the HFPA, which could also include memory blocks and additional programming layers.

Stansfield and Page[6] suggested a logic block built from an array of content-addressable memory cells, as opposed to LUT- or PLA-based blocks. The CAM cells can be used in RAM mode, in which case the logic block functions in the same way as a LUT. Also, multiple CAM cells can be combined to implement the equivalent functionality of a PLA. The study gives no comparison in terms of chip area or speed-performance between the CAM-based approach and a traditional FPGA or CPLD. However, limited information available from the project implies that the core block, a 4×4 CAM array, takes slightly more area than a 4-LUT. A PLA with 16 inputs and 8 pterms needs 10 of these arrays, more than the area of ten 4-LUTs. The HFPA can implement the same functionality in 40% of this area.

In related work from industry, Altera has recently introduced the APEX (Advanced Programmable Embedded Matrix) series of FPDs. Their key feature is the combination of LUTs and PLA-like blocks on the same chip, an idea similar to the one presented here. The APEX architecture contains embedded system blocks that can be configured to support pterms, CAMs, and memory blocks.[7] The products are fabricated in a 0.25-micron, five-layer metal process. APEX has a hierarchical interconnect structure similar to that of the Flex 10K, with the addition of a fourth level. The additional level is a direct result of FPD capacity increases. Other features include 100-MHz system performance and enhanced phase-locked loops. The first APEX devices will offer 500,000 gates, and future plans include devices of up to 2 million programmable gates.

References

1. J. Rose et al., "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, Oct. 1990, pp. 1217-1225.
2. J. L. Kouloheris and A. El Gamal, "PLA-Based FPGA Area vs. Cell Granularity," *Proc. 1992 Custom Integrated Circuits Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1992, pp. 4.3.1-4.3.4.
3. J. He and J. Rose, "Advantages of Heterogeneous Logic Block Architectures for FPGAs," *Proc. 1993 Custom Integrated Circuits Conf.*, IEEE CS Press, 1993, pp. 7.4.1-7.4.5.
4. S. Wilton, J. Rose, and Z. Vranesic, "Architecture of Centralized Field-Configurable Memory," *Proc. Third ACM Int'l Symp. Field-Programmable Gate Arrays*, ACM, New York, 1995, pp. 97-103.
5. E. Tau et al., "A First Generation FPGA Implementation," *Proc. Third Canadian Workshop on Field-Programmable Devices*, Polytechnical School of Montreal and University of Quebec at Montreal, 1995, pp. 138-143.
6. A. Stansfield and I. Page, "The Design of a New FPGA Architecture," *Proc. Fifth Int'l Workshop on Field-Programmable Logic and Applications*, Univ. of Oxford, London, 1995.
7. F. Heile and A. Leaver, *Proc. Seventh ACM Int'l Symp. FPGAs*, ACM, New York, 1999, pp. 13-16.

reason, we believe that the rest of the synthesis flow will be quite similar to synthesis for any architecture with a hierarchical routing structure. Therefore, synthesis to the HFPA is commercially viable, and the extra complexity is well justified by the new architecture's advantages.

Effects of technology-independent optimization. The first step in synthesizing circuits to a specific architecture is often technology-independent optimization. It consists of a series of partial collapsing and factoring (decomposition) operations. Decomposing the circuits efficiently is especially important when the target architecture accommodates only low fan-in nodes—for example, LUT-based FPGAs. The SIS optimization scripts may not be as vigorous as those in state-of-the-art, commercial logic synthesis tools, which may find better decompositions. Using these tools will lead to a lower area gain but a higher depth gain for the HFPA. The optimization algorithms in commercial tools are often integrated with specific target architectures. Thus, it is difficult to investigate the advantages of the HFPA with the technology-independent optimization methods incorporated in these tools. According to our comparison of script.rugged and script.algebraic, the latter finds better decompositions, leading to a lower area gain for the HFPA.

Another issue is that our choice of PALB was based on our analysis of benchmarks after optimization. If we had used different optimization tools, we might have reached slightly different results in terms of the PALB parameters. However, we believe that our current choices are reasonable.

IN THE FUTURE, we intend to investigate the HFPA synthesis issues more thoroughly. Our technology mapper needs some improvements. We also plan to develop a placement-and-routing tool to investigate the amount of routing resources and appropriate positioning of switches. Incorporating in the technology mapper an automated partial collapser designed especially for the HFPA might increase gain and make mapping easier.[7] Finally, it is likely that the HFPA can be enhanced in several ways, and we will continue improving the architecture. D&T

References

1. E.M. Sentovich et al., *SIS: A System for Sequential Circuit Synthesis*, Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley, 1992.
2. J. Rose et al., "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, Oct. 1990, pp. 1217-1225.
3. A. Aggarwal and D. Lewis, "Routing Architectures for Hierarchical Field-Programmable Gate Arrays," *Proc. Int'l Conf. Computer Design*, IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 475-478.
4. B.R. Owen et al., "BALLISTIC: An Analog Layout Language," *Proc. IEEE Custom Integrated Circuits Conf.*, IEEE CS Press, 1995, pp. 41-44.
5. J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table-Based FPGA Designs," *IEEE Trans. CAD of Integrated Circuits and Systems*, Vol. 13, No. 1, Jan. 1994, pp. 1-12.
6. H.J. Touati, H. Savoj, and R.K. Brayton, "Delay Optimization of Combinational Logic Circuits by Clustering and Partial Collapsing," *Proc. IEEE Conf. Computer-Aided Design*, IEEE CS Press, 1991, pp. 188-191.
7. A. Kaviani, *Novel Architectures and Synthesis Methods for High Capacity Field Programmable Devices*, PhD dissertation, Dept. of Electrical and Computer Engineering, Univ. of Toronto, 1999.
8. A.H. Farrahi and M. Sarrafzadeh, "Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping," *IEEE Trans. CAD of Integrated Circuits and Systems*, Vol. 13, No. 11, Nov. 1994, pp. 1319-1332.

Alireza Kaviani is a lecturer at the University of Toronto. His main research interests include architectures and synthesis methods for FPDs and microprocessor systems. He has more than three years of industry experience, including a year at Hewlett-Packard. Kaviani holds an MASc degree in computer engineering from the University of Toronto and a BS degree in electronic engineering from Sharif University, Iran. He received his PhD in electrical and computer engineering from the University of Toronto. He is a member of the IEEE and the Computer Society.

Stephen Brown is an associate professor of electrical and computer engineering at the University of Toronto. His dissertation on architecture and CAD for FPGAs won the Canadian NSERC's 1992 prize for the best doctoral thesis in Canada. He won a best paper award at the 1990 ICCAD. He has also won four awards for excellence in teaching electrical engineering, computer engineering, and computer science. He is a coauthor of the book *Field-Programmable Gate Arrays*. He was general and program chair of the Fourth Canadian Workshop on Field-Programmable Devices. Brown holds a PhD in electrical engineering from the University of Toronto. He is a member of the IEEE and the Computer Society.

Send questions and comments about this article to the authors at Dept. of Electrical and Computer Eng., 10 King's College Rd., University of Toronto, Toronto, Ontario, Canada M5S 3G4; kaviani@eecg.toronto.edu and brown@eecg.toronto.edu.