# Enhancing FPGA Performance for Arithmetic Circuits

Philip Brisk[1] Ajay K. Verma[1] Paolo Ienne[1]
[1]Processor Architecture Laboratory
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland
{philip.brisk, ajaykumar.verma, paolo.ienne}@epfl.ch

Hadi Parandeh-Afshar[1,2]
[2]Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
hparande@ut.ac.ir

## ABSTRACT
FPGAs offer flexibility and cost-effectiveness that ASICs cannot match; however, their performance is quite poor in comparison, especially for arithmetic dominated circuits. To address this issue, this paper introduces a novel reconfigurable lattice built from counters rather than look-up tables that can effectively accelerate the arithmetic portions of a circuit. We intend to integrate this novel lattice onto the same die as an FPGA.

## Categories and Subject Descriptors
B.7.1 [**Integrated Circuits**]: Types and Design Styles – *gate arrays*; B.2.4 [**Arithmetic and Logic Structures**]: High Speed Arithmetic – *algorithms, cost/performance.*

## General Terms
Algorithms, Performance, Design.

## Keywords
Field Programmable Gate Array (FPGA), Field Programmable Counter Array (FPCA) Look-up Table (LUT), Compressor Tree.

## 1. INTRODUCTION
The *Field Programmable Gate Array (FPGA)* is an attractive platform for hardware design due to its flexibility. FPGAs are often used for low-volume circuits that could not be profitably synthesized as an *Application Specific Integrated Circuit (ASIC)*. An off-the-shelf FPGA has already been fabricated and verified; these costs cannot otherwise be justified for low-volume designs. Compared to an FPGA implementation of a circuit, an ASIC reduces delay, area, and power consumption. These discrepancies are exacerbated for arithmetic circuits, which tend to be the ideal candidates for hardware rather than software implementations.

Originally, FPGAs were used to verify designs that would later be fabricated. Several iterations of *Moore's Law* through the mid-1990s allowed FPGAs to compete with ASICs in application domains, such as signal and video processing. FPGA vendors integrated *IP cores*—fixed, ASIC-like circuitry—into FPGAs. The discrepancy between ASICs and FPGAs remains, and to date, only incremental steps have been taken to narrow these gaps.

This paper addresses the discrepancy between FPGAs and ASICs by proposing a novel reconfigurable lattice that accelerates the arithmetic components of a circuit. This lattice, called a *Field Programmable Counter Array (FPCA)* can be integrated on the same die as an FPGA and presents itself as a programmable IP core to the designer. Unlike fixed IP cores, an FPCA is not limited to a specific application domain; likewise, the bitwidth of the operations that are mapped onto an FPCA are not fixed either.

## 2. RELATED WORK
An FPGA is a 2-dimensional array of *Configurable Logic Blocks (CLBs)*, which are connected to one another with a programmable interconnection network. A CLB contains a *k*-input, *Look-up Table (k-LUT)*, which can implement any *k*-input logic function. A CLB also includes a *flip-flop (FF)* (for sequential circuits) and a *multiplexer (mux)* that selects between the output of the LUT and the FF, as shown in Fig. 1. The selection bit of the mux (not shown) is set to configure the circuit as desired.

The high-end FPGAs which we consider in this paper are the *Xilinx Virtex-4* [14] and the *Altera Stratix-II* [4], both of which are realized in 90nm CMOS technology. In addition to LUTs, the *SLICEL* of the *Virtex-4* contains extra logic for fast addition. A fast carry-chain connects adjacent slices without connecting to the routing network. The *Stratix-II Adaptive Logic Module (ALM)* also has a fast carry-chain that can be used to implement a ripple-carry adder efficiently. In *Shared Arithmetic Mode*, the ALM can be configured as a ternary (3-input) adder.

Multi-input additions are often found in DSP applications. *Altera* [1] compared a 16-bit, 128-input adder tree on the *Stratix-II* and *Virtex-4*. The *Stratix-II* implementation required 5 levels of logic, 605 ALMs, and ran at 111.2 MHz; the *Virtex-4* implementation required 7 levels of logic, 1080 slices, and ran at 90.50 MHz, illustrating the advantage of ternary adders. The FPCA, described in Section 5, accelerates the performance of large summations.

## 3. COMPRESSORS AND COUNTERS
The best circuit to compute a multi-input addition is a *compressor tree*, which takes *k* inputs, $A_1$, …, $A_k$, of varying bitwidth, and produces two outputs *S (sum)* and *C (carry)* such that

$$S + C = \sum_{i=1}^{k} A_i \ . \qquad (1)$$

A compressor tree is built from *carry-save adders (CSAs)* instead of *carry-propagate adders (CPAs)*. The critical delay of each CPA is the path from carry-in to the carry-out. In a compressor tree a CPA is only needed to compute *S+C*. Compressor trees are also used in parallel multipliers [7, 11]. Readers unfamiliar with this material should consult a textbook on arithmetic [3, 6].

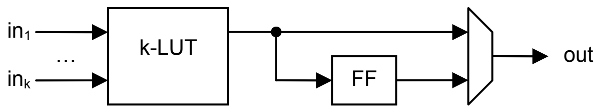

**Figure 1. Illustration of a k-input CLB.**

Many implementations of compressor trees exist. One approach is to built them from *m:n* counters, where *m* and *n* are the number of inputs and outputs respectively. A counter counts the inputs bits that are set to 1 (a value in the range *0..m*) and then outputs the number as a binary value, *n*. The number of output bits required is

$$n = \lceil \log_2(m+1) \rceil . \tag{2}$$

*2:2* and *3:2* counters are called *half-* and *full-adders* respectively. For *k' < k*, a *k*-input counter can implement a *k'*-input counter by setting *k – k'* input bits to 0. An optimal algorithm to construct a compressor tree from solely *3:2* and *2:2* counters was given by Stelling et al. [7]. Verma and Ienne [7] showed how to construct better counters from basic gates; they also presented an integer linear program that could construct an optimal compressor tree from a library of counters with sizes ranging from 2..k inputs.

Some circuits, for example FIR filters [5, 13], naturally contain large sums. In other cases, e.g. *Adpcm*, disjoint addition operations can be merged into a large sum by applying a set of transformations proposed by Verma and Ienne [9].

## 4. DSP/MAC BLOCKS ARE NOT ENOUGH

An initial question that motivated this work was whether IP cores could be used for large sums. We synthesized a few adder trees on the *Virtex-4* using both its general logic and the dedicated 18-bit adders in the *DSP48E* blocks [15]. In all cases, the implementation that used the *DSP48E* blocks had a slower frequency; the raw data has been omitted to conserve space.

## 5. FPCA ARCHITECTURE

An FPCA is similar to an FPGA, but with *m:n* counters replacing CLBs. The same interconnect and routing architecture is used for both. Similar to Fig. 1, a register and mux are placed on each counter output, and the selection bit of the mux is programmed. In an FPCA, a counter is called a *Fixed Counter Block (FCB).*

Since an FPCA is limited to sums, a hybrid FPGA/FPCA is proposed whereby the FPCA accelerates arithmetic computations. Fig. 2 illustrates an FPGA/FPCA and how to map a circuit onto it. We expect a hybrid device to contain multiple disjoint FPCAs interspersed throughout, not just one stripe, as shown in Fig. 2.

In Fig. 2, a circuit containing a large sum is shown on the left. The non-arithmetic portion is mapped onto the FPGA. The sum is divided into two parts: a compressor tree is mapped onto the FPCA; and the final addition *(S+C)* is mapped onto an adder. IThe adder can be implemented either as an IP core, or using a CLB with fast-carry logic. In our experimental evaluation, we assume that the FPGA performs the final addition.

IP cores are inherently limited because the bitwidth of each component is fixed. Although the counters in the FPCA are fixed, there is no principle limit on the bitwidth of the values that the FPCA can sum. Thus, the FPCA is much more flexible than an IP core in an FPGA. The primary drawback is that delays through the routing network are still present.

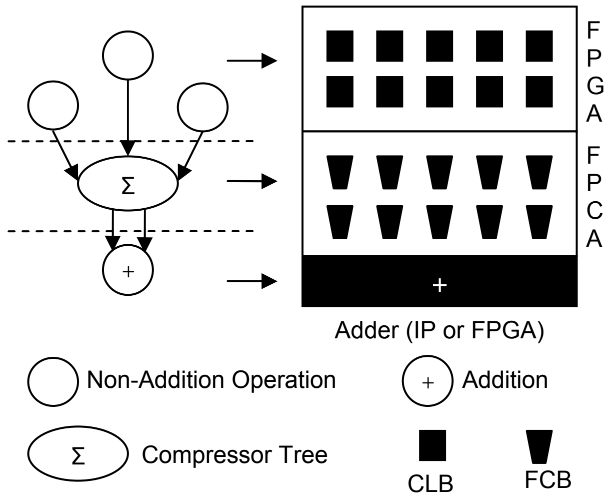

**Figure 2. Mapping a circuit with a large sum onto a hybrid FPGA/FPCA.**

### 5.1 Device Parameters

Several parameters of the FPCA are not yet determined. The dimensions of an FCB cannot exceed a CLB. Finding the best FCB structure is non-trivial, since there are many Pareto-optimal implementations of a circuit. For example, the largest, fastest 24-bit FCB might be too large, but the smallest, slowest 28-bit FCB might fit. Also related is the ideal size of the counter from the perspective of the application. For example, a 20-bit counter is not ideal for summation of 16 values. Our intention is to derive empirical answers through comprehensive benchmarking.

Other unresolved issues involve the distribution of CLBs and FCBs in a hybrid device. Pertinent questions include:

- What fraction of the blocks should be CLB and FCB?
- How many FPCAs should be placed in a hybrid device?
- What is the ideal aspect ratio for an FPCA; should FPCAs be placed as "stripes" or "islands" within the hybrid device?

At present, we do not have the answer to the aforementioned questions, nor have we identified the exact counter size. We are currently working on layout-related issues for both CLBs and FCBs in an attempt to realize a hybrid FPGA/FPCA in VLSI. We intend to answer these questions in a future publication.

### 5.2 CAD Flow

The basic stages of a CAD flow targeting an FPCA are:

1. Expose large sums as described by Verma and Ienne [9].
2. Given *k*, construct a compressor tree from *k*-input counters.
3. Place and route the circuit.

The FPCA is useless without the large sums found in Step 1. Step 2 subsumes *logic synthesis* and *technology mapping*; there is no need to optimize Boolean expressions since counters are used. For step 2, we build compressor trees from *k*-input counters using a method similar to Wallace's [11]. Step 3 uses FPGA placement and routing algorithms. Historically, simulated annealing has been used for placement [2]. Since our circuits have a specific structure, we hope to find an optimal polynomial-time algorithm or an efficient heuristic that produces comparable results to simulated annealing; this is future work.

## 6. EXPERIMENTAL RESULTS

We used *vpr* [2] to simulate the FPCA portion of a hybrid device and compare the results to commercial FPGAs. For comparison, we used the *Virtex-4* and *Stratix-II*, both of which are realized in 90nm CMOS technology. We configured *vpr* to simulate an FPCA with counters built using a 90nm standard cell library. We experimented with 4 counters, 8:4, 12:4, 16:5, and 20:5. Each counter was first built by the 3-greedy algorithm [7], converted to Reed-Muller form, and then optimized using the algorithm of Verma and Ienne [10]. The counters were then synthesized by *Synopsis Design Compiler* with *TSMC* 90nm standard cell components based on *Artisan* libraries. The area and delay of each counter is shown in Table 1. The delays are listed from the *most significant bit (MSB)* on the left to the *least (LSB)* on the right.

*vpr* simulates placement and routing on an FPGA. We configured the blocks to implement the 4 counters shown in Table 1.

To validate the efficacy of the FPCA, we present two case studies on real-world benchmarks are presented in Sections 6.1 and 6.2.

### 6.1 H.264/AVC Motion Estimation

H.264/AVC is presently the state-of-the-art for video coding [12]. The most challenging phase of this algorithm for hardware designers is the variable block size *motion estimation (ME)* phase. We implemented a *processing element (PE)* for H.264/AVC ME designed for a 1-D systolic array. Our implementation sums 5 values, four of which are 8-bit and one of which is 12-bit. We synthesized the datapath portion of this circuit onto the *Virtex-4* and *Stratix-II*, with and without an FPCA. Table 2 shows the result of this experiment. In Table 2, logic refers to the portion of the circuit that is always mapped onto the FPGA, and sum refers to the large summation—an adder tree for an FPGA, and a compressor tree plus final adder for a hybrid device. Together, these two components account for the total delay of the circuit.

For this experiment, we used a channel width of 40, which is the default for *vpr*, and set the aspect ratio to 1.

**Table 1. Counters realized in 90nm CMOS Technology.**

| Counter | Area (μm$^2$) | Delay (ns) (MSB…LSB) |
|---|---|---|
| 8:4 | 91.7 | (0.36, 0.38, 0.35, 0.25) |
| 12:4 | 175 | (0.37, 0.46, 0.35, 0.25) |
| 16:5 | 358 | (0.32, 0.62, 0.54, 0.49, 0.31) |
| 20:5 | 433 | (0.77, 0.78, 0.65, 0.48, 0.34) |

**Table 2. H.264/AVC Motion Estimation.**

| FPGA or Hybrid Device | Delay (ns) | | | Area |
|---|---|---|---|---|
| | Logic | Sum | Total | |
| Virtex-4 | 1.8 | 7.2 | 9.0 | 74 |
| +FPCA (8:4) | | 3.7 | 5.5 | 36+15 |
| +FPCA (12:4) | | 3.9 | 5.7 | 36+15 |
| +FPCA (16:5) | | 4.3 | 6.1 | 36+15 |
| +FPCA (20:5) | | 4.4 | 6.2 | 36+15 |
| Stratix-II | 1.3 | 3.9 | 5.2 | 74 |
| +FPCA (8:4) | | 3.0 | 4.0 | 39+15 |
| +FPCA (12:4) | | 3.2 | 4.2 | 39+15 |
| +FPCA (16:5) | | 3.6 | 4.6 | 39+15 |
| +FPCA (20:5) | | 3.7 | 4.7 | 39+15 |

Since only 5 values are summed, no advantage can be gained by increasing the number of bits for each HCB; in each case, a 8:4 counter provides the best results. Using this counter size, the critical path delay was reduced from 9.0ns to 5.5ns for the *Virtex-4*, and from 5.2ns to 4.0ns for the *Stratix-II*. The reason for the discrepancy between the two baseline FPGA implementations is that the *Stratix-II*'s ternary adders reduce the number of levels in the adder tree from 3 to 2; the *Virtex-4* requires 3 levels. In both cases, the FPCA effectively speeds up the circuit.

The area results in Table 1 are reported in terms of *LUT*s for the *Virtex-4* and *ALUTs* for the *Stratix-II*. For the hybrid devices, the results are reported in terms of *LUTS+FCBs* for the *Virtex-4* and *ALUTs+FCBs* for the *Stratix-II*. The total number of cells was reduced from 74, for both FPGAs, to 51 for the hybrid *Virtex-4/FPCA* and 54 for the hybrid *Stratix-II/FPCA*.

This result demonstrates the ability of an FPCA to speed up circuits that compute relatively small sums. For larger sums, a much greater impact is expected.

### 6.2 FIR Filters

*Finite Impulse Respose (FIR) Filters* [5, 13] are common DSP operations that have successfully been accelerated on FPGAs. A FIR filter typically contains one very large summation operation, in the context of constant multiplication by a set of coefficients. Here, we evaluate the performance of three different FIR filters: 6-tap, 10-tap, and 20-tap on an FPCA. To conserve space, results are only presented for 8:4 and 12:4 counters.

FIR filters are typically pipelined in order to achieve high throughput, and there is a well-known tradeoff between the number of pipeline stages (latency) and timing (delay) of the design. For the hybrid device, we chose to pursue aggressive pipelining, and used the register on the output of every FCB. In principle, retiming could be used to reduce the number of pipeline stages by increasing the latency of the design; work in this area is beyond the scope of this paper.

We built the FIR filters using distributed arithmetic [13], which has been used in the past to reduce area for FPGAs. Tables 3 and 4 show the results of our experiments with the *Virtex-4* and *Stratix-II* respectively.

In order to achieve routability with *vpr*, we increased the channel width to 80 and fixed the dimensions of the FPCA to 20×20 for the 6- and 10-tap filters. This was too small for the 20-tap filter, so we increased the dimensions to 30×30. Once again, the aspect ratio was set to 1. In Section 6.1, we allowed *vpr* to select the default dimension of the FPCA that could fit the design while maintaining the aspect ratio.

In each case, experiment, the hybrid FPGA/FPCA ran faster than the FPGA alone. For the hybrid device, the FPGA performed the final addition. It is difficult to contrast the delays of each circuit, because the number of pipeline stages differed. In a few cases, such as the 6-tap filter on both devices, we achieved a lower delay with fewer pipeline stages. This testifies to the appropriateness of counters as building blocks for large sums, rather than CPAs.

Also notable, the hybrid device significantly reduced the number of blocks required to compute the sum, by a complete order of magnitude in the case of both the 10- and 20-tap filters.

**Table 3. FIR Filters for Stratix-II and Hybrid Device.**

| | Taps | Delay (ns) | Area | Pipeline Stages |
|---|---|---|---|---|
| Stratix-II | 6 | 3.1 | 944 | 7 |
| +FPCA (8:4) | | 3.0 | 97+165 | 5 |
| +FPCA (12:4) | | 3.0 | 97+142 | 3 |
| Stratix-II | 10 | 3.8 | 1454 | 7 |
| +FPCA (8:4) | | 3.2 | 150+205 | 8 |
| +FPCA (12:4) | | 3.0 | 150+126 | 4 |
| Stratix-II | 20 | 3.5 | 2665 | 7 |
| +FPCA (8:4) | | 3.2 | 265+398 | 10 |
| +FPCA (12:4) | | 3.4 | 265+280 | 7 |

**Table 4. FIR Filters for Virtex-4 and Hybrid Device.**

| | Taps | Delay (ns) | Area | Pipeline Stages |
|---|---|---|---|---|
| Virtex-4 | 6 | 5.1 | 721 | 7 |
| +FPCA (8:4) | | 3.8 | 96+165 | 5 |
| +FPCA (12:4) | | 3.8 | 96+142 | 3 |
| Virtex-4 | 10 | 5.7 | 1284 | 8 |
| +FPCA (8:4) | | 4.0 | 149+205 | 8 |
| +FPCA (12:4) | | 3.8 | 149+126 | 4 |
| Virtex-4 | 20 | 5.2 | 2945 | 9 |
| +FPCA (8:4) | | 4.0 | 264+398 | 10 |
| +FPCA (12:4) | | 4.2 | 264+280 | 7 |

### 6.3 Discussion

It is important to note that the observed delays vary considerably, according to the routing. For relatively sparse designs, *vpr* tended to place the FCBs toward the edges of the device, placing the bulk of the white space in the middle. Often, the critical delay of the design was a single net routed across the white space to the other side of the chip. Different routing algorithms, it seems, might have different results. For the results shown in Section 6.2, we switched from *vpr*'s default routing algorithm to bounding box routing, and the delays observed were reduced considerably.

In general, the delay through the routing network was 2-3× greater than the combinational delays through the circuit. It is important to note that *vpr* is an academic tool that is almost 10 years old, and that the software used to program an FPGA is commercial and recent. We acknowledge the fact that different routing algorithms have been used for the FPGA and FPCA portions of these circuits and that the commercial routing tools use the current state-of-the-art; the same is true for critical path delay estimates.

## 7. CONCLUSION

An FPCA has been introduced to accelerate the performance of arithmetic computations on FPGAs. The FPCA accelerates the computation of large multi-input additions, which can be exposed in a circuit by applying transformations proposed by Verma and Ienne [9]. The FPCA accelerates these sums by mapping them onto a 2-dimensional lattice of counters, which can be viewed as a programmable Wallace-like [11] compressor tree. Previous studies have shown that the current MAC/DSP blocks embedded in FPGAs cannot accelerate these types of computations sufficiently. The intention is to integrate an FPGA and an FPCA onto the same die, using the FPCA for arithmetic acceleration. Our experiments with H.264/AVC Motion Estimation and several FIR filters demonstrate that a hybrid device FPCA can effectively accelerate these computations.

In the future, we intend to answer some of the architectural questions raised in Section 5.1, and to further develop the CAD flow described in Section 5.2. We also intend to develop a model to analyze the power consumption of an FPCA, and to work toward a VLSI realization of a hybrid FPGA/FPCA.

## ACKNOWLEDGMENTS

We wish to acknowledge insightful comments made by Jason Brown and Laura Pozzi. Theo Kluter provided timely assistance with the synthesis of the counters in Table 1.

## REFERENCES

[1] Altera Corporation. *Stratix II vs. Virtex-4 Performance Comparison, v. 2.0*. White paper. September, 2006.

[2] Betz, V., Rose, J., and Marquardt, A. *Architecture and CAD for Deep Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[3] Ercegovac, M., D., and Lang, T. *Digital Arithmetic*, Morgan-Kauffman, San Francisco, CA, USA, 2004.

[4] Lewis, D., et al. The Stratix II logic and routing architecture. *Int. Symp. Field Prog. Gate Arrays (FPGA '05)* (Monterey, CA, USA, February 20-22, 2005) 14-20.

[5] Mirzaei, S., Hosangadi, A., and Kastner, R. FPGA implementation of high speed fir filters using add and shift method. *Int. Conf. Computed Design (ICCD '06)*, (San Jose, CA, USA, October 1-4, 2006)

[6] Parhami, B. *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, NY, USA, 2000.

[7] Stelling, P. F., Martel, C. U., Oklobdzija, V. G., and Ravi, R. Optimal circuits for parallel multipliers. *IEEE Trans. Computers, 47, 3* (March, 1998), 273-285.

[8] Verma, A. K., and Ienne, P. Automatic synthesis of compressor trees: reevaluating large counters. *Design Automation and Test in Europe (DATE '07)* (Nice, France, April 16-20, 2007).

[9] Verma, A. K., and Ienne, P. Improved use of the carry-save representation for the synthesis of complex arithmetic circuits. *Int. Conf. Computer-Aided Design (ICCAD '04)* (San Jose, CA, USA, November 7-11, 2004) 791-798

[10] Verma, A. K., and Ienne, P. Improving XOR-dominated arithmetic circuits by exploiting dependencies between operands. *Asia and South Pacific Design Automation Conf. (ASP-DAC '07)* (Yokohama, Japan, January 23-26, 2007)

[11] Wallace, C. S. A suggestion for a fast multiplier, *IEEE Trans. Electronic Computers, 13*, 1964 14-17

[12] Wiegand, T., Sullivan, G. J., Bjøntegaard, G., and Luthra, A. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits and Systems for Video Tech., 13, 7* (July, 2003) 560-576

[13] Xilinx Corporation. *Distributed Arithmetic FIR Filter v. 9.0*, Product Specification. 2004.

[14] Xilinx Corporation. *Virtex-4 User Guide, v. 2.1*. White paper.

[15] Xilinx Corporation. *XtremeDSP for Virtex-4 FPGAs User Guide, v. 2.4*. White paper.