Architectural Improvements for Field Programmable Counter Arrays: Enabling Efficient Synthesis of Fast Compressor Trees on FPGAs

Alessandro Cevrero[1,2], Panagiotis Athanasopoulos[1,2], Hadi Parandeh-Afshar[2], Ajay K. Verma[2], Philip Brisk[2], Frank K. Gurkaynak[1], Yusuf Leblebici[1], Paolo Ienne[2]

[1]Microelectronic Systems Laboratory
Institute of Microelectronics and Microsystems
Ecole Polytechnique Federale de Lausanne (EPFL)
Lausanne, Switzerland, CH-1015

[2]Processor Architecture Laboratory
School of Computer and Communications Sciences
Ecole Polytechnique Federale de Lausanne (EPFL)
Lausanne, Switzerland, CH-1015

{first_name.last_name}@epfl.ch

ABSTRACT

The Field Programmable Counter Array (FPCA) was introduced to improve FPGA performance for arithmetic circuits. An FPCA is a reconfigurable IP core that can be integrated into an FPGA. To exploit the FPCA, a circuit is transformed by merging disparate addition and multiplication operations into large multi-input addition operations, which are synthesized as compressor trees on the FPCA; the remaining portion of the circuit is synthesized on the FPGA. This paper presents a series of architectural improvements to the FPCA that reduce routing delay, increase flexibility and component utilization, and simplify the integration process. Using an FPGA containing six FPCAs, we observed average and maximum speedups of 1.60× and 2.40× on a set of arithmetic benchmarks.

Categories and Subject Descriptors

B.6.1 [Logic Design]: Design Styles–FPGAs; B.2.4 [Arithmetic and Logic Structures]: High-Speed Arithmetic–cost/performance

General Terms: Design, Performance.

Keywords: FPGA, Field Programmable Counter Array (FPCA).

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) offer many advantages compared to *Application Specific Integrated Circuits (ASICs)*, including reduced non-recurring engineering costs, post-deployment reconfigurablity, and reduced time-to-market. The cost for a typical mask set to fabricate an ASIC using 45nm CMOS technology runs in excess of $1,000,000. A designer, on the other hand, can purchase an off-the-shelf FPGA (in 65nm or 90nm CMOS technology, for now) and program it for a miniscule fraction of the cost. The resulting circuit, however, will be slower, consume more power, and utilize significantly more silicon resources than its ASIC equivalent. These gaps are significant, but tolerable, for finite state machines and control-dominated circuits, but become more pronounced for arithmetic-dominated circuits.

To address this discrepancy, Brisk et al. [5] introduced the *Field Programmable Counter Array (FPCA)*, a reconfigurable IP core that accelerates multi-operand addition, which occurs in parallel multipliers [8, 23] and applications such as video coding [6], FIR filters [15], and 3G wireless base station channel cards [18]. Arithmetic transformations [22] can also expose large multi-operand additions in arithmetic circuits. Using these transformations, we propose to map an arithmetic circuit onto a hybrid FPGA/FPCA, where the compressor tree is synthesized onto the FPCA, and the remaining portions onto the FPGA.

This paper presents a series of improvements to the original FPCA architecture. The most important features include counters of varying size and flexibility, hardwired connections between counters (replacing a programmable FPGA-like routing network), and an integrated carry-propagate adder; furthermore, the new architecture simplifies the mapping and integration processes. Compared to prior work on *Generalized Parallel Counter (GPC)* mapping [16] (which is faster than using ternary adder trees), we observed speedups of as much as 2.40×, and 1.60× on average.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'08, February 24–26, 2008, Monterey, California, USA.
Copyright 2008 ACM 978-1-59593-934-0/08/02...$5.00.

2. RELATED WORK

To improve arithmetic performance, several researchers proposed carry chains that could efficiently embed circuitry that could perform fast addition inside a series of adjacent logic blocks [7, 9, 10, 12, 14]. Carry chains have been adopted by commercial vendors: The Xilinx Virtex-4/5 CLBs can send propagate/generate signals to adjacent blocks [26, 27]; the Altera Stratix II/III *Adaptiv Logic Modules (ALMs)* implement ripple-carry addition [4-6]. In the Stratix II ALM, Altera introduced support for ternary, addition using the carry-chains [1, 2]. The *Look-Up Tables (LUTs)* act as *3:2* compressors, and the carry chain adds the result; a similar idea was incorporated into the Xilinx Virtex-5 [27].

Hard IP cores, e.g., DSP/MAC blocks, have been embedded into FPGAs [28]. Kastner et al. [11] developed a technique to profile a set of applications to identify commonly occurring operation patterns, yielding domain-specific FPGAs. Kuon and Rose [13] warned that the benefits of IP cores could be lost due to mismatches in bitwidth; the FPCA imposes no such restrictions.

Verma and Ienne [22] proposed circuit transformations that fuse disparate addition and multiplication operations into compressor trees. Poldre and Tammemae [17] synthesized *4:2* compressors [26] on Xilinx Virtex FPGAs; however, no commercial tools, to our knowledge, use their solution. Parandeh-Afshar et al. [16] also developed techniques to synthesize compressor trees on FPGAs using 6-input GPCs (modern FPGAs have 6-input LUTs), achieving a considerable speedup over ternary adder trees.

Wang et al. [24] replaced some programmable wires in the FPGA routing fabric with *HArdwired Routing Patterns (HARPs)*, which reduced delay and power consumption, but limit flexibility. The FPCA architectures described here attempt to use HARPs in a more systematic fashion, which is feasible because the application domain is limited to compressor trees.

3. PRELIMINARIES

3.1 Compressor Trees

A compressor tree [23] is a circuit that adds $k > 2$ n-bit binary integers, $A_0, \ldots, A_{k-1}$, where $A_i = (a_{i,n-1}, \ldots, a_{i,0})$, for $0 \le i \le k$. The critical path delay of a compressor tree is much less than the delay of an adder tree, built from *Carry-Propagate Adders (CPAs)*. To compute the result a compressor tree produces values, *Sum (S)* and *Carry (C)*, where the final sum, $S+C$ is computed by a CPA:

$$S + C = \sum_{i=0}^{k-1} A_i \ . \qquad (1)$$

The *rank* of a bit is its subscript index describing its position in the integer, e.g., bit $a_{i,r}$ has rank r. The *Least Significant Bit (LSB)* has rank *0* and the *Most Significant Bit (MSB)* has rank *k-1*. Bit $a_{i,r}$ of rank r represents quantity $a_{i,r} \times 2^r$. A *column* $C_r = \{a_{0,r}, \ldots, a_{k-1,r}\}$ is the set of input bits of rank r. The input to a compressor tree is often viewed as a set of columns, rather than integers.

3.2 Single-Column Counters

A *Single Column (m:n) Counter* is a circuit that takes m input bits, counts the number of bits that are set to *1*, and produces the sum as an n bit value. In adder design, *2:2* and *3:2* counters are called *half* and *full* adders respectively; a parallel array of disconnected *3:2* counters can be referred to as a *Carry-Save Adder (CSA)*. For a fixed value of m, the number of output bits required is:

$$n = \lceil log_2(m+1) \rceil . \qquad (2)$$

Wallace [23], Dadda [8], and Stelling et al. [19], and others, have systematically built compressor trees from CSA; Verma and Ienne [21] used larger counters, ranging from *2:2* to *8:4*.

In the FPCA, an *m:n* counter can implement an *m':n* counter, provided that $m' \le m$ (note that n may exceed the number of bits required to represent a value in the range *[0, m']*). To support this functionality, the *FPCA-1.2* architecture introduces an *Input Configuration Circuit (ICC)* which allows any of the m inputs to be set to *0*. A single-bit ICC is shown in Fig. 1(a).

3.3 Generalized Parallel Counters

A *Generalized Parallel Counter (GPC)* [20] is an extension of an *m:n* counter that can count input bits of multiple ranks. A GPC is specified as a tuple: $G = (m_{k-1}, \ldots, m_0; n)$, where the counter takes m_i inputs of rank i, $0 \le i \le k-1$, and sums them; otherwise, the functionality of a GPC is the same as that of an *m:n* counter *c*.

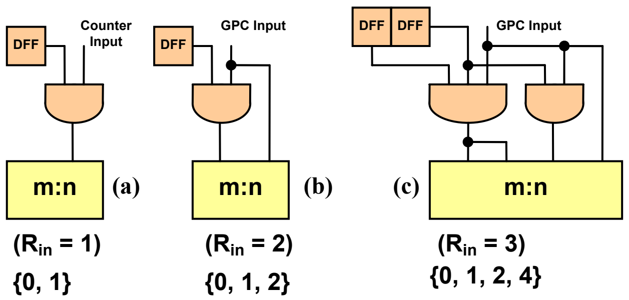


Figure 1.

Counter Input Configuration Circuit (a); GPCs with R_{in} = 2 (b) and 3 (c)

Let $M = m_0 + \ldots + m_{k-1}$ be the number of GPC inputs. In an FPCA, the size of the GPC is limited by M, which we will assume to be fixed. Let b_i be a bit of rank i. b_i contributes a value of $b_i \times 2^i$ to the total sum of bits. An *m:n* counter can count b_i by connecting it to 2^i inputs. An *m:n* counter can implement the functionality of an M-bit GPC g, provided that:

$$m \ge \sum_{i=0}^{k-1} m_i 2^i \qquad (3)$$

In the FPCA, both M and m are constant for GPCs, and $m > M$. Thus, an *m:n* counter can implement many different GPCs.

A *Configurable GPC* is an *m:n* counter preceded by a *GPC Configuration Circuit (GPCCC)*, which is programmed to implement a variety of M-input GPCs. Let b_i be an input to the *m:n* counter. The *rank of the input*, R_{in}, is defined to be the maximum rank any bit can take. $R_{in} = 1$ for an *m:n* counter, as shown in Fig. 1(a). The GPCCC circuits for $R_{in} = 2$ and *3* are shown in Fig. 1(b) and (c) respectively. The *D Flip-Flops (DFFs)* control the rank of each incoming bit, and are programmed when the user configures the FPCA. The allowable input values (including 0, when the input bit is '0') for each bit for different R_{in} values are shown in Fig. 1 as well.

4. FPCA ARCHITECTURE: OVERVIEW

This section introduces the *FPCA-1.0* architecture [5] and its four successors: *FPCA-1.1*, *1.2*, *1.3*, and *2.0*. The basic unit of computation for an FPCA is called a *Compressor Slice (CSlice)*.

4.1 The FPCA-1.0 Architecture

The *FPCA-1* architecture is similar to an FPGA but with LUTs replaced by *m:n* counters. Except for selection between the counter output and register, the CSlice is not configurable. In a hybrid FPGA/FPCA, both devices share the same global routing network; the FPCA replaces LUTs with counters in one (or more) rectangular subregions. Thus, the *FPCA-1.0* architecture can implement a compressor tree with significantly fewer logic levels than any type of compressor synthesized on an FPGA.

From ASIC design, we have strong evidence that *m:n* counters are the clear choice for compressor tree synthesis [22]. Modern FPGAs have 6-input LUTs, and thus, they cannot implement a counter or GPC containing more than 6 inputs within one level of logic. To the best of our knowledge, no 6-input compressor has been proposed to date which can utilize either the carry chains of either Xilinx or Altera. *FPCA-1.0*, meanwhile, places no limit on the size of the counters that comprise their CSlices.

4.2 CSlice Design and Integration

One of the key challenges of FPGA design is to balance the active area used by the logic circuit (LUT, ALM, etc.), the resources required for configuration, and the area required by the input and output connections to the logic block. To implement the FPCA structure, we envision a system where the CSlice occupies more or less similar area to the FPGA primitive (ALM or LUT). In this way (and assuming several border issues can be resolved within reasonable effort) we foresee an architecture where CSlices replace several Logic slices of the FPGA.

By design, FPCA CSlices reduce a large number of inputs to a smaller number of outputs. The first level counter determines the number of incoming connections to a CSlice. While increasing the counter size adds some complexity to the active circuit area of the FPCA, it increases the number of inputs at an even higher rate. Our goal is to strike a balance between the number of inputs to the FPCA CSlice and the amount of active area.

Our preliminary investigations have assumed a limit of 16 inputs per physical tile that can be occupied by either an FPGA primitive (ALM or LUT) or an FPCA CSlice. For *FPCA-1.2*, this input constraint directly determines the first level counter size (*15:4*) and thereby the active area occupied by the entire CSlice. Note that the input limitation here is simply a parameter; similar results will be obtained with any other number as well. For this particular restriction we observed that the CSlice occupied only about half the circuit area of a comparable FPGA primitive. From this observation it was clear that a more efficient architecture can be designed, if larger counters can be utilized within the tile without increasing the number of inputs. This has led to the development of *FPCA-1.3* where the main difference is that a *31:5* counter is used at the core of a configurable GPC that allows 16 inputs to be mapped to the 31 available counter inputs.

The design of the *FPCA* CSlice, starting with *FPCA-1.1*, was motivated by prior work on HARPs in FPGAs [24]. A HARP is a direct connection in the routing network that bypasses switch boxes at routing intersections. Using a HARP instead of a programmable wire reduces wire delay and power dissipation; however, the inclusion of HARPs in the routing fabric reduces flexibility. Since the application domain of FPCAs is limited to compressor trees, we felt that that *FPCA-1.0* would be much more amenable to HARPs than a traditional FPGA.

By examining the structure of compressor trees, we found a regular interconnection pattern, if we assume that all columns have m bits. Fig. 2 shows an example where $m = 15$; the basic interconnection structure, which compresses a single column, is shown on the right. A *15:4* counter, produces a sum bit of rank i in column i, and propagates carry bits of increasing rank to columns $i+1$, $i+2$, and $i+3$. After the first level comprised of *15:4* counters, all columns at the second level will have four bits, so *4:3* counters are used; all columns at the third level will have 3 bits, so *3:2* counters are used. At the fourth level, 2 bits remain per column, so a CPA sums the result. Circuitry to implement this pattern is shown on the right-hand-side of Fig. 2.

This yields the following pattern: given a contiguous series of columns of m bits, an $m{:}n$ counter will produce columns of n bits, at the subsequent level (ignoring the boundaries). By recursively applying this pattern, we can generate the pattern for any value m. Table 1 shows the number of levels and the counter sizes required to replicate this pattern for different values of m.

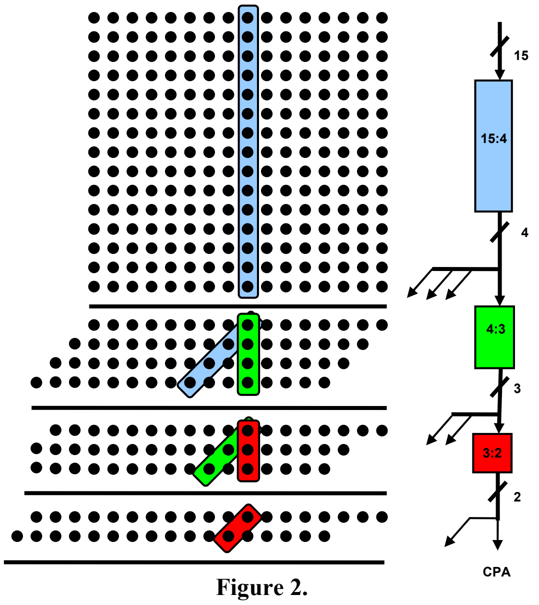


Figure 2.

Compressor tree using counters of different sizes

Table 1.

Levels and counter size inside a CSlice.

m	Levels	Counters, CPA
$0 \le m \le 7$	2 + CPA	(m:3), (3:2), CPA
$8 \le m \le 15$	3 + CPA	(m:4), (4:3), (3:2), CPA
$16 \le m \le 31$	3 + CPA	(m:5), (5:3), (3:2), CPA
$32 \le m \le 63$	3 + CPA	(m:6), (6:3), (3:2), CPA
$64 \le m \le 127$	3 + CPA	(m:7), (7:3), (3:2), CPA

Some variation of the circuit shown in Fig. 2 is the skeletal structure of every CSlice, beginning with *FPCA-1.1*.

An FPCA is a set of CSlices, $S = \{S_0, S_1, \ldots, S_{k-1}\}$, where each CSlice S_i compresses m bits of rank i. Fig. 3 shows an example of interconnected *CSlices* for $m = 15$. CSlice S_i propagates carry-bits produced by its local counters and CPA to CSlices S_{i+1}, S_{i+2}, and S_{i+3}. Such an FPCA can compute the sum of up to k columns, with at most m bits per column (km bits, in total). In Fig. 3, $k = 4$ and $m = 15$, so the FPCA can compress as many as 60 bits.

4.3 CSlice Architecture: Evolution

Fig. 4 shows the different FPCA CSlice architectures. The following sections of the paper didactically describe the evolution from *FPCA-1.0* (a) to *FPCA-2.0* (e).

FPCA-1.1 introduces the pattern of descending counters and hardwired connections described in Section 4.2; *FPCA-1.2* eliminates the routing network altogether, replacing it with local connections; *FPCA-1.3* increases the size of the *m:n* counter, and adds a layer of GPC configuration to make it flexible; and the *FPCA-2.0* CSlice is able to compress multiple columns at once.

5. THE FPCA-1.1 CSLICE

The use of *m:n* counters in the *FPCA-1.0* architecture does not eliminate the routing delay between counters. As process geometries shrink into the deep submicron scale, the critical paths in the routing fabric will become dominant, since wires do not scale as well as transistors.

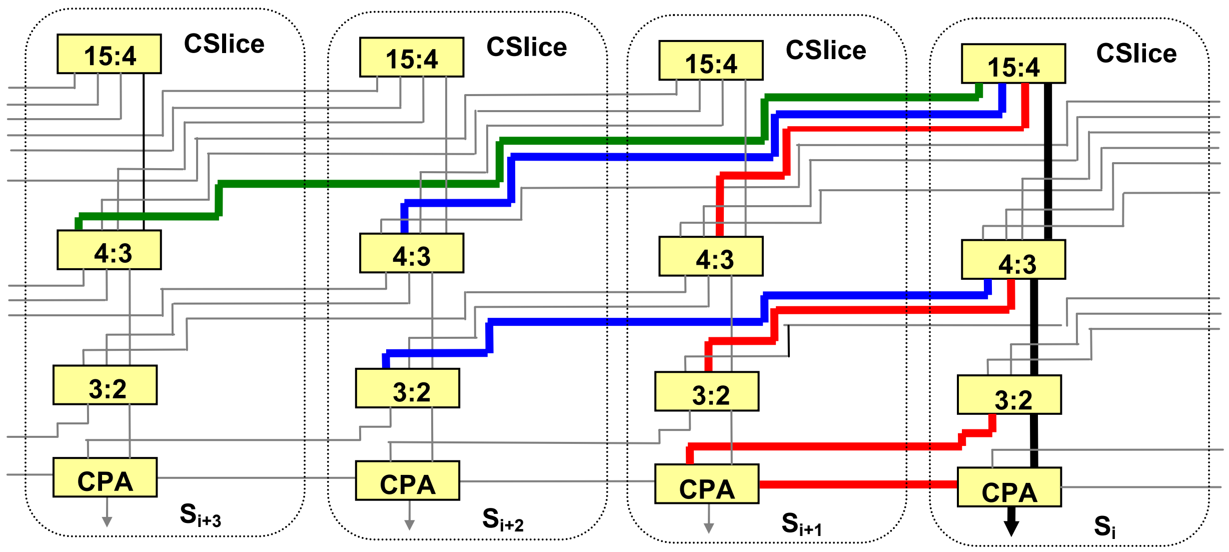


Figure 3.

Cascading CSlices to propagate carry-bits.

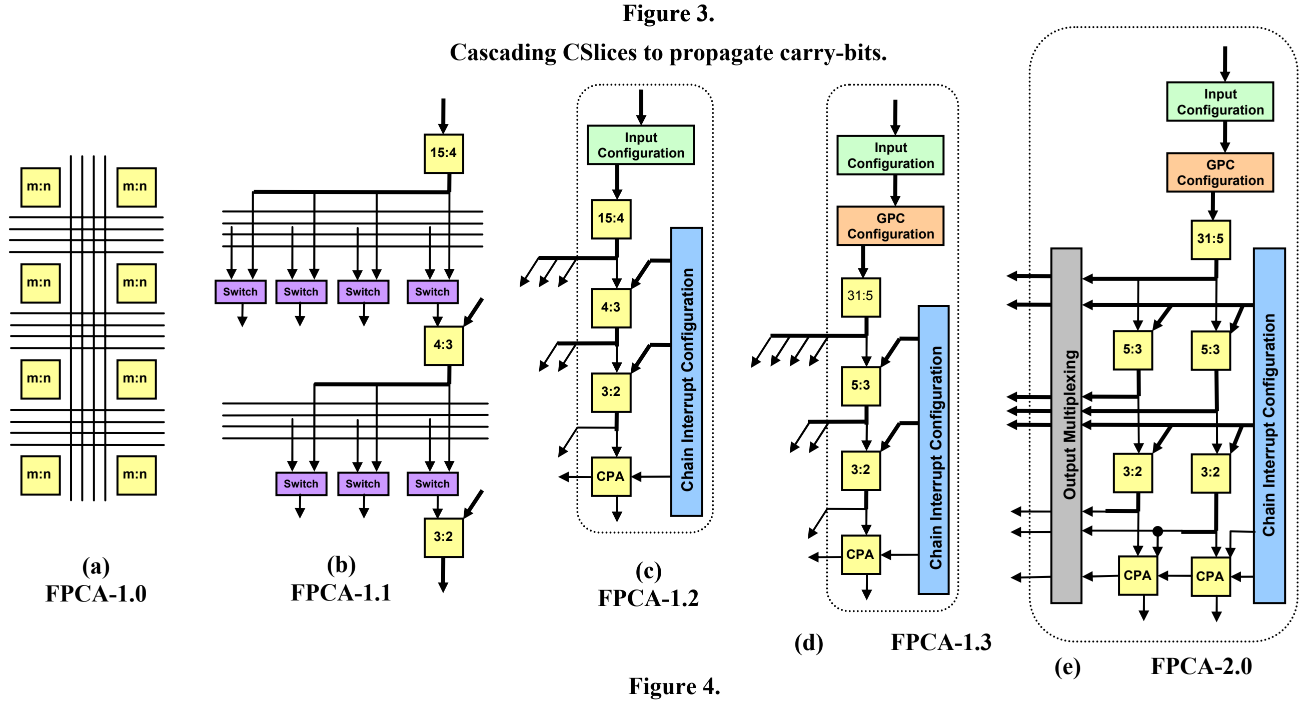


Figure 4.

Evolution of the FPCA/CSlice architecture.

The *FPCA-1.1* architecture, shown in Fig. 4(b), was motivated by the pattern shown in Fig. 2. The result was the organization of a set of counters of decreasing size into a CSlice, according to Table 1, with HARPs placed between counters in the same CSlice. The carry bits propagated from one CSlice to the next, shown in Fig. 3, are reminiscent of carry-chains used in FPGA logic cells for fast arithmetic [7, 9, 10, 12, 14]. In this architecture, a switch is placed on all HARPs between counters. The switch determines whether the preceding counter or an input taken from the horizontal routing network connects to each counter input. If there are only three or four bits in a column, for example, the switch allows direct access to the *3:2* and *4:3* counters. Although these switches add delay to each HARP, the delay is deterministic, unlike delays through the routing network.

Furthermore, the critical delays within each CSlice are not dependent on the efficacy of the placement and routing algorithms used to program the device.

6. THE FPCA-1.2 CSlice

The *FPCA-1.2* CSlice, shown in Fig. 4(c), was introduced to eliminate the horizontal routing channels in *FPCA-1.1*. The interface between the FPGA and FPCA becomes similar to the boundary between FPGA logic and IP cores, rather than similar lattices with differences in planar geometry and channel width.

The programmable switch between counters has been removed from the *FPCA-1.2* CSlice. This reduces the critical path delay in the CSlice, but at the cost of some flexibility, as direct access to the smaller counters is no longer provided. In Fig. 4(c), a column of four bits is summed using a *15:4* counter, with eleven input bits set to '0.' One possibility would be to route the '0' bits from the FPGA to the FPCA; however, doing this would consume previous routing resources in the FPGA, that would be better allocated for other purposes. Instead, an ICC (Section 2.2) has been placed immediately before the counter. The ICC is programmed by the user to propagate either a CSlice input or the value '0' to each input of the *m:n* counter within the CSlice.

When a *15:4* counter is configured to implement a *4:3* counter, many internal (and possibly external) signals are driven to '0,' thus reducing critical delay. Although the delay is still more than the delay of a *4:3* counter, we are not particularly concerned about this, because an effective mapping algorithm will map bits from multiple columns onto the slice in question, in addition to the four bits in the current column. Thus the *15:4* counter is more likely to be utilized as a GPC than to be wholly underutilized.

Each *FPCA-1.2* CSlice also has an integrated CPA: a ripple-carry adder, similar to the carry-chain in Altera Stratix II/III FPGAs [1, 3]. A more sophisticated adder, such as a parallel prefix adder [10], would cause the CSlices to become non-uniform, which would complicate the layout of the circuit. In *FPCA-1.0* and *1.1*, it was assumed that the final addition would be performed on the FPGA's general logic, using the carry chains. Integrating the CPA into the *FPCA-1.2* CSlice eliminates one layer of routing delay to transport the bits from the FPCA to the FPGA and reduces the overall number of bits to transmit.

The *FPCA-1.2* CSlice also includes a *Chain Interrupt Configuration Circuit (CICC)*, which permits multiple compressor trees to be synthesized on an FPCA, as long as there are a sufficient number of CSlices available. The CICC is programmed to pass the carry-out bits from the preceding CSlice to the current CSlice, or to drive all carry-in bits to '0,' effectively isolating two adjacent CSlices from one another. The CICC requires one configuration bit per CSlice, and one 2-input AND gate per incoming wire. If the configuration bit is '0,' then the chain is interrupted, otherwise, the bits propagate into the CSlice.

7. THE FPCA-1.3 CSLICE

The *FPCA-1.2* architecture is well-suited for rectangular bit patterns where all columns have fifteen or fewer bits; however, it does not perform particularly well for irregular bit patterns where the number of bits in consecutive columns is different. Fig. 5 shows an irregular bit pattern, derived from a 3-tap FIR filter. The number of bits per column varies from one to nineteen.

The *15:4* counter in CSlice S_i can count fifteen bits of rank i, but at most seven of rank $i+1$ and three of rank $i+2$. Thus, when an *m:n* counter is configured as a GPC in the *FPCA-1.2* CSlice, the CSlice inputs are dramatically underutilized. This impedes the potential performance of the *FPCA-1.2* architecture for irregular bit patterns, such as Fig. 5.

As stated in Section 4.2, the number of CSlice inputs is a limiting factor, not the area of the CSlice. We found that we could increase the size of the counter to *31:5* without exceeding our area budget for the *FPCA-1.3* CSlice, which is shown in Fig. 4(d). We also added an extra input port, for sixteen in total. The *31:5* counter is used to implement a 16-input configurable GPC, using a GPCCC, as described in Section 2.3. Since the number of *m:n* counter inputs now exceeds the input capacity of a CSlice, the CSlice input utilization is higher. The GPC, for example, can sum fifteen rank-$(i+1)$ bits by connecting each bit to two counter inputs.

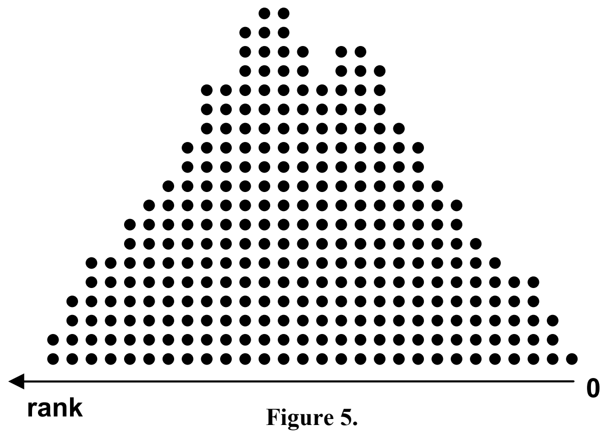
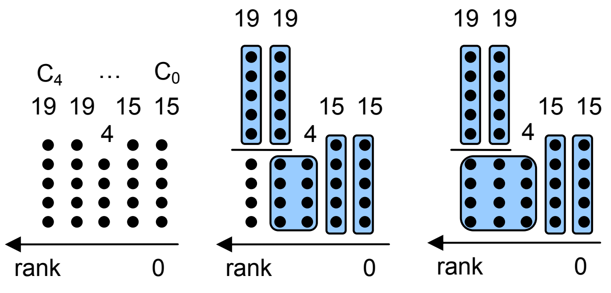


Figure 5.

Irregular input bit pattern for a 3-tap FIR filter.



(a) (b) (c)

Figure 6.

Columns of bits to be summed (a); the last 4 bits cannot be mapped to an FPCA-1.2 CSlice (b); FPCA-1.3 can accommodate all of the columns (c).

We have developed *FPCA-1.3* CSlices with two different GPCCCs, that supports GPCs with maximum input rank 2 or 3. The former is smaller in terms of area and complexity, but the latter allows for a greater number of GPCs to be implemented. The GPC configuration is determined when the device is programmed.

Fig. 6(a) shows an irregular pattern of bits that illustrates the advantages of the *FPCA-1.3* CSlice architecture over *FPCA-1.2*. The input is a set of 5 columns, $C = \{C_0, \ldots C_4\}$, where C_i is the number of bits of rank i: $C_0 = C_1 = 15$, $C_2 = 4$, and $C_3 = C_4 = 19$.

In Fig. 6(b) and (c), we attempt to map the columns onto an FPCA comprising 5 *FPCA-1.2* and *1.3* CSlices: $S = \{S_0, \ldots, S_4\}$. In both cases, the first two columns, C_0 and C_1 map directly onto CSlices S_0 and S_1; likewise, 15 of the 19 bits in columns C_3 and C_4 map directly onto CSlices S_3 and S_4. This leaves us with 12 bits—four bits per column, from columns C_2, C_3, and C_4 that must map onto slice S_2.

The *FPCA-1.2* CSlice can only accommodate the remaining bits from columns C_2 and C_3, but not C_4. The ranked sum of the bits from C_2 and C_3 is $4 \times 2^0 + 4 \times 2^1 = 12 < 15$, so the *15:4* counter in the *FPCA-1.2* architecture can accommodate them. Even if one bit from column C_4 was taken, the ranked sum would increase to 16, beyond the capacity of the counter. Now, if all 12 bits were accepted, the ranked sum becomes $12 + 4 \times 2^2 = 26$. Since the *FPCA-1.3* CSlice contains a *31:5* counter at its core, sufficient bandwidth is available if $R_{in} = 3$.

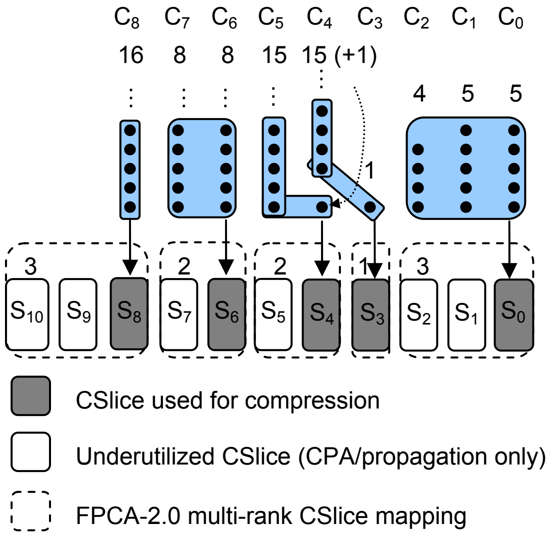
8. THE FPCA-2.0 ARCHITECTURE

The use of GPCs in the *FPCA-1.3* architecture can lead to the underutilization of whole CSlices. Fig. 7 shows an example where 8 columns, $C = \{C_0, ..., C_8\}$, ranging from five to nineteen bits per column, are mapped onto eleven CSlices, $S = \{S_0, ..., S_{10}\}$.

C_0, C_1, and C_2 contain *5*, *5*, and *4* bits respectively. All of these bits can be mapped onto S_0 since $5{\times}2^0 + 5{\times}2^1 + 4{\times}2^2 = 31$. Next, we group the single bit in C_3 with *15* (of *16* total) bits from C_4. These bits can also be mapped onto a CSlice since $1{\times}2^0 + 15{\times}2^1 = 31$; the bits, however, cannot be mapped onto S_1 or S_2. These two slices propagate the carry-out bits produced by the counters/CPA in S_0. Some of these bits will arrive at the CPAs in S_1 and S_2. Mapping the bits from columns C_3 and C_4 to CSlices S_1 and/or S_2 would effectively reduce the rank of bits, and the FPCA would produce an incorrect result. Therefore, no bits can be mapped onto S_1 or S_2 in this example. In this case, the *31:5* counter, the largest component in the *FPCA-1.3* CSlice, is not used in these CSlices; thus, we classify S_1 and S_2 as underutilized.

In Fig. 7, six of the eleven slices are underutilized. In the case of S_9 and S_{10}, underutilization is unavoidable because summing numbers inevitably produces carry-bits, beyond the rank of the most significant bit of the input.

The *FPCA-2.0* architecture, shown in Fig. 4(e), addresses the issue of underutilization. Prior CSlices could produce one output bit. The *FPCA-2.0* CSlice, in contrast, operates on the granularity of words rather than bits. Producing bits of multiple output ranks provides an extra degree of freedom to the mapping algorithm, which can use these configurations to reduce the number of CSlices used. Each CSlice retains a single *31:5* counter, but the remaining portions of the compressor tree are replicated two or three times, permitting the computation of multiple sum bits within a CSlice. The area cost of the replicated portions of the compressor tree (including CPAs) is negligible compared to the area of the *31:5* counter. Each CSlice has *rank-1*, *2*, and *3* configurations, and can produce 1, 2, or 3 output bits in parallel.



C8 C7 C6 C5 C4 C3 C2 C1 C0
16 8 8 15 15 (+1)
4 5 5
1
3 2 2 1 3
S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
CSlice used for compression
Underutilized CSlice (CPA/propagation only)
FPCA-2.0 multi-rank CSlice mapping

Figure 7.

FPCA-1.3 underutilizes CSlices; FPCA-2.0 solves this shortcoming with multi-rank CSlice configurations.

The following CSlices in Fig. 7 are replaced with a single slice in the *FPCA-2.0* architecture: $\{S_0, S_1, S_2\}$ – rank *3*; $\{S_3\}$ – rank *1*; $\{S_4, S_5\}$ – rank *2*; $\{S_6, S_7\}$ – rank 2; $\{S_8, S_9, S_{10}\}$ – rank 3. Using the *FPCA-1.3* CSlice, Fig. 7 would employ eleven *31:5* counters, six of which are unused; switching to *FPCA-2.0*, there would be a total of five *31:5* counters, all of which would be used. Thus, the multiple rank configurations of the *FPCA-2.0* CSlice enable better utilization of the *31:5* counters, which are the most expensive resource in a CSlice in terms of area.

Two more modifications to the CSlice are necessary to support multiple rank configurations. The first is an output multiplexing stage that can drive the correct signals to the neighboring CSlice, depending on the configuration. The second is a CPA that can be configured to produce 1, 2, or 3 bits of output. The CPA in each slice is a carry-select adder, where the number and size of the adder stages are programmable. The CPA is only programmed after a rank configuration has been established for each CSlice.

8.1 Configurable Carry-Select Adder

In a carry-select adder, the bits are partitioned into *m* groups, where group *i* contains P_i bits; here, we partition CSlices rather than bits, so group *i* contains P_i CSlices. Groups are chosen when the FPCA is programmed. The first CSlice in a group performs a different function than the others in the same group. Therefore, each CSlice is designed to implement both functions.

The first slice in a group must be able to assume an incoming carry of *0* or *1* from the previous group, and select the correct sum value accordingly. The remaining slices within a group must propagate the carry from the previous slices, while also selecting the correct sum. Fig. 8 shows the CPA adder for one CSlice.

Within a CSlice, there are two 3-bit ripple-carry adders and a multiplexer to select between the output of each FA in the ripple-carry-adder, depending on when the slice is configured to be rank-1, 2 or 3. The output of one of the two ripple-carry adders is selected, depending on the value of the carry-in. The DFF in Fig.8 is set if the CSlice is the first in its group. The different behaviors are realized via the multiplexers on the right-hand-side of Fig. 8.

9. MULTI-FPCA CONFIGURATIONS

More than one FPCA may be necessary to implement a large compressor tree. This requires that several FPCAs are located relatively close to one another within a larger FPGA.

Horizontal configurations (Fig. 9(a)) occur when there are more columns to be summed than the number of CSlices on an FPGA. The interconnection structure remains the same as Fig. 3; however, the global routing network must be used to connect the carry-outputs of the last CSlice in the first FPCA to the carry-inputs of the first CSlice in the second; another possibility, which we may investigate in the future, is to use HARPs.

Vertical configurations (Fig. 9(b)) occur when the number of bits per-column exceeds the capacity of one CSlice. If *m* is the capacity of a CSlice, suppose that each column has *km* bits. Then *k* CSlices (e.g. *k* FPCAs) are needed to compress each column; this will result in *k* sum bits produced per-column, one by each FPCA. Another FPCA is now required to sum the remaining bits.

The main advantage of a compressor tree compared to an adder tree is that a CPA is only needed to perform the final addition. A vertical multi-FPCA configuration, however, uses a CPA at each level of the tree. This is unavoidable in the CSlice architectures shown in Fig. 4, because there is no way to bypass the CPA.

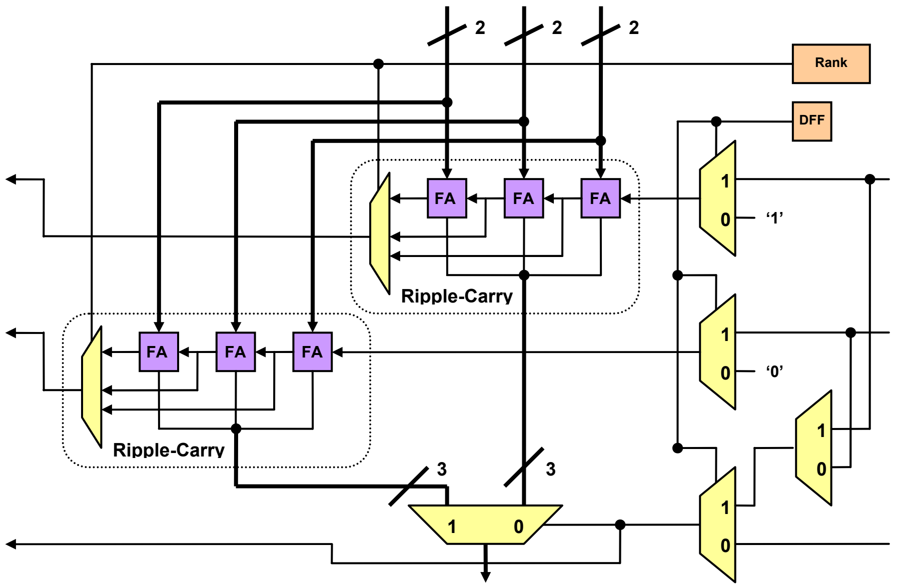


Figure 8.

Configurable carry-select adder for FPCA-2.0 CSlice.

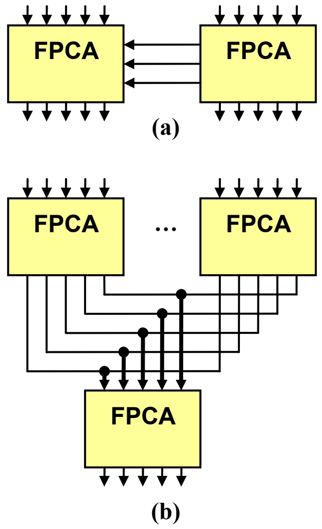


Figure 9.

Horizontal (a) and vertical (b) multi-FPCA configurations

To allow bypassing to support vertical configurations, we have added additional CSlice outputs, which are connected to the *sum* and *carry* bits produced by each 3:2 counter. The user can select these outputs, instead of the *sum* outputs of the CPA, to bypass the CPA to reduce the critical path delay of a large compressor tree; however, each FPCA will produce twice as many outputs using this configuration, which increases the demand for routing resources.

10. FPCA MAPPING HEURISTIC

Here, we introduce the problem of mapping columns onto FPCAs, focusing on the *FPCA-2.0* CSlice architecture. The goal of the problem is to minimize the height of the compressor tree. Although the discussion focuses on single FPCA configurations, the approach described here easily generalizes to multi-FPCA configurations: if there are more columns than slices per FPCA, then horizontal configurations are needed. If there are unmapped bits, then a vertical configuration is required: the unmapped bits are combined with the FPCA outputs, and the resulting columns are mapped onto a FPCA. Given these extensions, the remaining portions of this section focus on mapping onto a single FPCA.

10.1 Problem Formulation

Let $B = \{b_0, \ldots, b_{M-1}\}$ be a set of bits to sum, where $rank(b)$ is the rank of bit $b \in B$. The bits are organized into k columns: $C = \{C_0, \ldots, C_{k-1}\}$; $C_i = \{b \in B \mid rank(b) = i\}$ is the set of bits of rank i. The bits in B are ordered so that for each pair of bits, b_j and b_{j+1}, $rank(b_j) \leq rank(b_{j+1})$. Thus, the first $|C_0|$ bits belong to C_0, the next $|C_1|$ bits belong to C_1, etc.

The target device is an instance of the *FPCA-2.0* architecture, comprised of k CSlices: $S = \{S_0, \ldots, S_{k-1}\}$. The problem remains the same if there are fewer columns than CSlices. We are also given: R_{in} (Section 2.3), which limits the possible GPC configurations, and is the same for all CSlices; and N, the number of input connections to each CSlice.

The output is a function $f\colon B \rightarrow \{\perp, 0, 1, .., k-1\}$ that describes the mapping of bits onto CSlices. For bit $b \in B$, $f(b) = \perp$ if b is not mapped to a CSlice; otherwise, $f(b) = i$, $0 \leq i \leq k-1$.

Let $B_i = \{b \in B \mid f(b) = i\}$ is the set of bits mapped onto CSlice S_i; $B_\perp = \{b \in B \mid f(b) = \perp\}$ is the set of unmapped bits.

For each bit b_j we define the quantity Δ_j as follows:

$$\Delta_j = \begin{cases} 0 & \text{if } b_j \in B_\perp \\ rank(b_j) - f(b_j) & \text{if } b_j \in B - B_\perp \end{cases} \qquad (4)$$

A legal mapping solution satisfies the following two constraints:

$$|B_i| \leq N \qquad 0 \leq i \leq k-1, \qquad (5)$$

$$0 \leq \Delta_j \leq R_{in} - 1 \qquad 0 \leq j \leq M-1, \text{ and} \qquad (6)$$

Constraint (5) ensures that the number of bits assigned to a CSlice does not exceed N, the number of CSlice input ports. Constraint (6) ensures that the rank of a bit must not be smaller than the rank of a CSlice. Clearly, we can map a bit of larger rank to a CSlice of smaller rank by connecting the bit to multiple counter inputs (as permitted by the GPCCC); however, we cannot map a bit of smaller rank to a CSlice of larger rank by connecting it to less than one input. For example, a CSlice S_2 (of rank 2) counts values $0, 4, 8, 12, \ldots$; the CSlice does not have sufficient granularity to count all the values of a rank-1 bit: $0, 2, 4, 6, 8, \ldots$, or a rank-*0* bit: *0, 1, 2, 3, …*. Constraint (6) also ensures that the difference between the rank of each bit b_j mapped onto CSlice S_i does not exceed $R_{in} - 1$. For example, if $R_{in} = 3$, then S_i can only take bits from three columns: C_i, C_{i+1}, C_{i+2}, or equivalently, bits whose ranks are i, $i+1$, or $i+2$.

The optimal solution is the one that minimizes the height of a compressor tree built from FPCAs with vertical connections.

10.2 Mapping Heuristic

Here, we describe a heuristic for the FPCA mapping problem described in the previous section. We have not yet analyzed the complexity of the problem; it may or may not be NP-Complete.

The input to the problem is twofold: a set of columns of bits to be added, and a library of GPCs containing the GPCs that can be implemented by each CSlice. Our FPCA CSlice has a *31:5* counter at its core, $R_{in} = 3$ meaning that it can compress up to 3 columns at once, and $N = 16$ CSlice inputs. We found 58 GPC configurations that could be supported by these constraints.

The first step of the heuristic is to cover all of the input columns with GPCs. We used our prior heuristic for GPC mapping [15] for this purpose, restricting the input library of available GPCs to the 58 described above. The next step is to map the groups of covered bits onto CSlices in one (or more) FPCAs.

Let $G = \{G_1, \ldots, G_P\}$ be the covering, i.e., each GPC G_i contains at most 16 bits spanning 3 columns. Limiting the number of bits per GPC satisfies Constraint (5), since each GPC will be mapped onto one CSlice. The limit of 3 columns per GPC ensures that all GPCs satisfy Constraint (6).

The *rank* of a GPC, $R(G_i) = min\{rank(b) | b \in G_i\}$ is the minimum rank among all bits in G_i. Let K be the number of CSlices in the FPCA ($K = 8$ in our experiments). We must pack the GPCs found by the covering into groups of at most K GPCs, such that each group can be mapped onto an FPCA.

The packing process must satisfy the following constraints: (i) if $R(G_i) = R(G_j)$ then G_i and G_j cannot be mapped onto the same CSlice (or packed into the same FPCA); (ii) if $R(G_j) > R(G_i)$, G_i and G_j can be packed into consecutive CSlices in the same FPCA only if $1 \leq R(G_j) - R(G_j) \leq 3$.

To pack the different GPCs in the covering onto FPCAs, we find chains of GPCs that satisfy constraint (ii) above. If we find such a chain that contains more than K GPCs, the chain can only be realized with horizontal configurations between multiple FPCAs, as described in Section 9. After each chain is identified, the GPCs in the chain are removed; then the process repeats and a new chain among the remaining GPCs is found. The process stops after all GPCs have been mapped to an FPCA CSlice.

In the example of Fig. 7, there is a single chain of five GPCs, which are mapped directly onto the CSlices (shown with dashed lines); rank-*1*, *2*, and *3* configurations are all used. Likewise, the example of Fig. 6(c) has a single chain of five CSlices as well.

Two (or more) short chains can be mapped onto an FPCA by breaking the carry propagation with the CICC (Section 6). After the initial mapping phase, we look for pairs of short chains that can utilize unused CSlices on FPCAs that have already been allocated. An example of this is shown in Fig. 10.

The rank-configuration of each CSlice is determined based on which GPC is mapped onto it. If the GPC is an *m:n* counter, then a single-rank configuration is appropriate; otherwise, a rank-2 or rank-3 configuration is selected. Based on the configuration, the appropriate CSlice outputs are selected and then generate the bits remaining in each column following the first layer of compression.

If there is at most 1 bit per column, we are done; otherwise, we need another layer of compression and a vertical configuration, as described in Section 9, is required. In this case, we configure each CSlice in the previous level to produce 2 outputs per column from the 3:2 compressors, so that we bypass the CPAs. Then we repeat the mapping process for the resulting columns of bits.

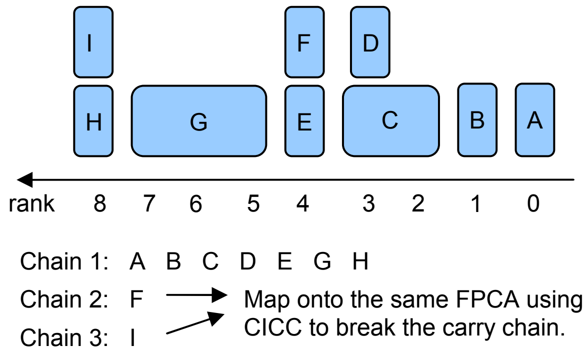


Figure 10.

Example of packing GPCs into FPCAs. 3 chains are found, two of which can be mapped onto the same FPCA.

Table 2.

CSlice Synthesis Results

CSlice	rank-1	rank-2	rank-3
Area [μm^2]	1240	2347	2770
Delay [ns]	0.40	0.71	0.73
CPA Delay [ns]	0.04	0.05	0.07
Dyn. Power [mW]	1.44	1.46	1.85

11. EXPERIMENTAL RESULTS

11.1 FPCA Synthesis and Verification

We wrote a VHDL description of an FPCA using the *FPCA-2.0* CSlice architecture. We limited the number of CSlices per FPCA to eight, so that an FPCA would have approximately the same width as a DSP block in a traditional FPGA. We believe that having dimensions similar to an established IP core would simplify the integration process. The configuration bitstream was implemented with an array of DFFs. In practice, commercial FPGAs use SRAM cells, which are smaller; we chose DFFs to avoid the effort required for a full custom design. We verified the correctness of both models using *Mentor Graphics Modelsim v6.1*; synthesis was performed using *Synopsys Design Compiler* and *Design Vision*; and placement and routing was performed with *Cadence Design Systems*' *Silicon Encounter*. The design kit used was a 90nm *Artisan* standard cell library.

We designed and verified three versions of the CSlice, featuring rank-*1*, rank-*2*, and rank-*3* configurations. The synthesis results for each CSlice are shown in Table 2; only the rank-3 CSlice was used in our experiments. Although the rank-3 CSlice is the slowest in terms of delay, it spans 3 columns; thus, the delay through one instance of the rank-3 CSlice is less than the delay of three rank-1 CSlices concatenated to one another.

11.2 Single-FPCA Delay Extraction

It is challenging to analyze the delay of an FPCA (or FPGA) without first programming it, due to false paths and loops. To extract the delay, we synthesized each benchmark on the FPCA as described in Section 10.2. We then configured the FPCA to perform the desired functionality and re-synthesized, placed, and routed the design with all configuration bits set (we instructed the synthesizer not to propagate and optimize the "constant" values). This gave us a good estimate of the critical path delay for each benchmark when it is actually mapped onto the FPCA.

11.3 Multi-FPCA Delay Extraction

The methodology for delay extraction outlined in the preceding section does not account for routing delays when multiple FPCAs are used. We extracted the actual routing delay from an Altera Stratix II FPGA using an approach outlined in this Section.

We defined a pre-placed soft IP core whose dimensions correspond to an FPCA. Let F* be the function implemented by the core (some trial and error was required to find an appropriate function that would yield the desired area). It should be noted that F* was defined to have the same number of inputs and outputs as an 8-CSlice FPCA. F* was written in VHDL and mapped, synthesized, placed, and routed onto the *Stratix II FPGA* by *Altera*'s *Quartus II Software*. This gave us a reasonable estimate of the critical path delay along each path of F* on our soft core. We pre-placed instances of F* on our FPGA in 2 columns of 3; this mimics our intended placement of FPCAs.

If the mapping heuristic from Section 10.2 produced multi-FPCA configurations, we generated a VHDL description of the system, but replaced each FPCA instance with an instance of F* instead. We synthesized the resulting circuit onto the FPGA using the pre-placed instances of F*. Through manual analysis of the results, we extracted the routing delays between instances of F*, as well as delays between each instance of F* and I/O pins. We then combined these routing delays with the combinational delays extracted for each FPCA, as described in Section 11.2. Short of fabricating our own device, we believe that this is the most accurate delay measurement that we could achieve using the tools at hand; we strongly believe that this methodology is more accurate than the use of a simulator, such as *VPR* [4].

11.4 Results

The evaluation of the FPCA focuses on arithmetic benchmarks. *fir3* and *fir6* are 3- and 6-tap FIR filters [8, 15]; *m12x12* and *m16x16* are parallel integer multipliers. *ME* is one *Processing Element (PE)* of an internally developed systolic array architecture for the motion estimation phase of *H.264/AVC* video coding. *ME* uses a compressor tree to aggregate *Sum-of-Absolute-Difference (SAD)* computations; *mac* is a multiply-accumulator. The other benchmarks are arithmetic circuits that have been transformed to expose compressor trees by Verma and Ienne [22].

The baseline approach to compressor tree synthesis is the GPC mapping heuristic of Parandeh-Afshar et al. [16]; in their study, GPC mapping synthesized compressor trees with significantly less delay than ternary adder trees, the previous state-of-the-art. The GPC mapping heuristic targeted an Altera Stratix II FPGA. The FPCA mapping targeted a similar device with 6 FPCAs; the delay was extracted using methods described in the preceding section.

Fig. 11 shows the results of the experiment. The critical path delays of the circuits are normalized to the critical path delay of GPC mapping. The normalized delay for GPC mapping is 1, and the reduction in critical path delay using the FPCA is reported as a speedup relative to GPC mapping. On average, the speedup observed was 1.60×. The largest speedups were observed for *ME* (2.40×) and *mac* (2.37×). For the other benchmarks, the speedups ranged from 1.20× (*fir6)* to 1.72× (*fir3*).

Table 3 shows the number of logic levels and the number of resources (FPCAs, LABs) consumed by each benchmark when synthesized on FPCAs and using GPC mapping, respectively. Table 3 also lists the bitwidth of the compressor tree output.

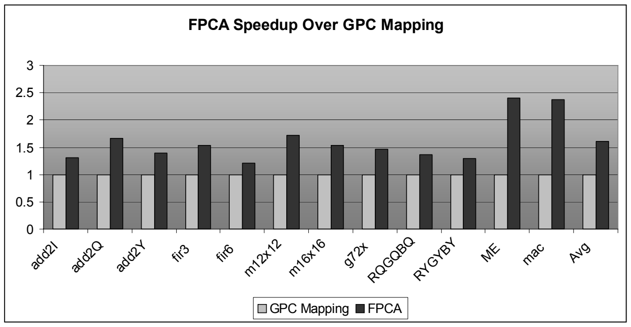


Figure 11.

Speedup of using an FPCA (with rank-3 CSlices) compared to GPC Mapping.

Table 3.

Number of logic levels and the resources used (area), for compressor trees synthesized using FPCAs and GPC Mapping; also, the output bitwidth of each compressor tree.

Benchmark	Levels		Resources		Output Bitwidth
	FPCA	GPC	FPCAs	LABs	
add2I	2	4	8	31	30
add2Q	1	3	2	16	30
add2Y	2	4	6	39	30
fir3	2	3	6	21	29
fir6	2	4	8	42	31
m12x12	1	2	2	10	25
m16x16	1	3	3	20	32
g72x	1	2	3	17	40
RQGQBQ	2	3	4	12	17
RYGYBY	2	3	4	13	17
ME	1	2	1	7	13
mac	1	2	1	6	16

Table 3 shows that FPCAs has fewer logic levels than GPC mapping for all benchmarks, and considerably fewer FPCAs than LABs were used, echoing a similar resource utilization result reported by Brisk et al. [5] for the *FPCA-1.0* architecture. The largest speedups were observed for *ME* and *mac*, the benchmarks that used the fewest LABs when synthesized via GPC mapping, and also had the smallest output bitwidth.

To some extent, the horizontal and vertical configurations for each benchmark can be inferred from Table 3; however, the precise organization cannot be inferred without knowing the pattern of input bits. From the input bit pattern, we can infer, from the mapping heuristic of Section 10.2, the rank configuration of each CSlice. For example, the input to *g72x* was eight 32-bit numbers (e.g., 32 columns of 8 bits). This required 40 output bits, the maximum among all benchmarks, as shown. Each CSlice was configured as rank-2, and consumed 16 input bits (every bit from each pair of columns). Altogether, this required three FPCAs, organized in a horizontal configuration. The critical path delay includes the combinational delay through the counters in each CSlice, but 40 bits of CPA; due to the horizontal configurations, there are also two instances of routing delay between subsequence FPCAs. This routing delay could be reduced, or wholly eliminated, in principle, if the number of CSlices per FPCA was increased.

12. CONCLUSION AND FUTURE WORK

This paper has introduced several architectural improvements for FPCAs, including hardwired connections between counters, counters of multiple sizes, GPCs, fast carry chains between CSlices, and CSlices containing multiple rank configurations. Experimentally, we observed speedups of as much as 2.40×, in terms of combinational delay, compared to synthesis using GPC mapping [16]; the average speedup was 1.60×.

We envision several different avenues for future work. The most important is to study the integration of the FPCA into an FPGA. Kuon and Rose [13] have already argued that the cost of routing data to and from IP cores significantly diminishes their impact on performance; the FPCA itself is a special case of this, with a particularly high I/O bandwidth requirement compared to other IP cores of similar size. We also intend to investigate pipelined versions of the FPCA that could increase throughput. Lastly, we intend to study new structures for the CPA, possibly based on carry-lookahead addition, that lead to reduced delay.

REFERENCES

[1] Altera Corporation, *Stratix II Device Handbook, vol. 1* and *2*, available online: http://www.altera.com/

[2] Altera Corporation, *Stratix II vs. Virtex-4 Performance Comparison*, available online: http://www.altera.com/

[3] Altera Corporation, *Stratix III Device Handbook, vol. 1* and *2*, available online: http://www.altera.com/

[4] Betz, V., Rose, J., and Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*, Springer, 1999.

[5] Brisk, P., Verma, A. K., Ienne, P., and Parandeh-Afshar, H. Enhancing FPGA performance for arithmetic circuit, *Design Automation Conf. (DAC '07)* (San Diego, CA, USA, June 4-8, 2007) 334-337.

[6] Chen, C-Y., Chien, S-Y., Huang, Y-W., Chen, T-C., Wang, T-C., and Chen, L-G. Analysis and architecture design of variable block-size motion estimation for H.264/AVC, *IEEE Trans. Circuits and Systems-I, vol. 53, no. 2*, February, 2006, 578-593.

[7] Cherepacha, D., and Lewis, D. DP-FPGA: an FPGA architecture optimized for datapaths. *VLSI Design, vol. 4, no. 4*, 1996, 329-343.

[8] Dadda, L., Some schemes for parallel multipliers, *Alta Frequenza, vol. 34*, May, 1965, 349-356.

[9] Frederick, M. T., and Somani, A. K. Multi-bit carry chains for high-performance reconfigurable fabrics. *Int. Conf. Field Prog. Logic and Applications (FPL '06)* (Madrid, Spain, August 28-30, 2006) 1-6.

[10] Hauck, S., Hosler, M. M., and Fry, T. W. High-performance carry chains for FPGAs, *IEEE Trans. VLSI Systems, vol. 8, no. 2*, April, 2000, 138-147.

[11] Kastner, R., Kaplan, A., Ogrenci-Memik, S., and Bozorgzadeh, E. Instruction generation for hybrid reconfigurable systems. *ACM Trans. Design Automation of Electronic Systems, vol. 7, no. 4,* October, 2002, 605-627.

[12] Kaviani, A., Vranseic, D., and Brown, S. Computational field programmable architecture, *IEEE Custom Integrated Circuits, Conf. (CICC '98)* (Santa Clara, CA, USA, May 11-14, 1998) 261-264.

[13] Kuon, I., and Rose, J. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design, vol. 26, no. 2*, February, 2007, 203-215.

[14] Leijten-Nowak, K., and van Meerbergen, J. L., An FPGA architecture with enhanced datapath functionality, *Int. Symp. FPGAs (FPGA '03)* (Monterey, CA, USA, February 23-25, 2003) 195-204.

[15] Mirzaei, S., Hosangadi, A., and Kastner, R. High speed FIR filter implementation using add and shift method, *Int. Conf. Computer Design (ICCD '06)* (San Jose, CA, USA, October 1-4, 2006).

[16] Parandeh-Afshar, H., Brisk, P., and Ienne, P. Efficient synthesis of compressor trees on FPGAs. *Asia and South Pacific Design Automation Conf. (ASPDAC '08)* (Seoul, Korea, January 21-24, 2008).

[17] Poldre, J., Tammemae, K. Reconfigurable multiplier for Virtex FPGA family, *Int. Workshop on Field- Programmable Logic and Applications (FPL '99)* (Glasgow, UK, August 30 – September 1, 1999) 359-364.

[18] Sriram, S., Brown, K., Defosseux, R., Moerman, F., Paviot, O., Sundararajan, V., and Gatherer, A. A 64 channel programmable receiver chip for 3G wireless infrastructure, *IEEE Custom Integrated Circuits Conf. (CICC '05)* (San Jose, CA, USA, September 18-21, 2005) 59-62.

[19] Stelling, P. F., Martel, C. U., Oklobdzija, V. J., and Ravi, R. Optimal circuits for parallel multipliers, *IEEE Trans. Computers*, *vol. 47, no. 3*, March 1998, 273-285.

[20] Stenzel, W. J., Kubitz, W. J., and Garcia, G. H. A compact high-speed parallel multiplication scheme, *IEEE Trans. Computers, vol. C-26, no. 10,* October, 1977 948-957.

[21] 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) 443-448.

[22] 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.

[23] Wallace, C. S. A suggestion for a fast multiplier, *IEEE Trans. Elec. Computers, vol. 13*, February, 1964, 14-17.

[24] Wang, G., Sivaswamy, S., Ababei, C., Bazargan, K., Kastner, R., and Bozorgzadeh, E. Statistical analysis and design of HARP FPGAs, *IEEE Trans. Computer-Aided Design, vol. 25, no. 10*, 2006, 2088-2102.

[25] Weinberger, A. 4:2 carry-save adder module, *IBM Technical Disclosure Bulletin, vol. 23*, Jan. 1981.

[26] Xilinx Corporation, *Virtex-4 User Guide*, available online: http://www.xilinx.com/

[27] Xilinx Corporation, *Virtex-5 User Guide*, available online: http://www.xilinx.com/

[28] Zuchowski, P. S., Reynolds, C. B., Grupp, R. J., Davis, S. G., Cremen, B., and Troxel, B. A hybrid ASIC and FPGA architecture, *Int. Conf. Computer-Aided Design (ICCAD '02)* (San Jose, CA, USA, November 10-14, 2002) 187-194.