An FPGA Architecture with Enhanced Datapath Functionality

Katarzyna Leijten-Nowak[1,2]
[1]Eindhoven University of Technology
Design Automation Section
Den Dolech 2
Eindhoven, The Netherlands
knowak@natlab.research.philips.com

Jef L. van Meerbergen[2,1]
[2]Philips Research Labs
Embedded System Architectures on Silicon
Prof. Holstlaan 4, WDC3
Eindhoven, The Netherlands
jef.van.meerbergen@philips.com

ABSTRACT

Although FPGAs are a cost-efficient alternative for both ASICs and general purpose processors, they still result in designs which are more than an order of magnitude more costly and slower than their equivalents implemented in dedicated logic. This efficiency gap makes FPGAs less suitable for high-volume cost-sensitive applications (e.g. embedded systems).

We show that the intrinsic cost of traditional general-purpose FPGAs can be reduced if they are designed to target an application domain or a class of applications only. We propose a method of the application-domain characterization and apply it to characterize DSP. A novel FPGA logic block architecture derived based on such an analysis, and which exploits properties of target applications, is presented. Its key feature is the 'mixed-level granularity' being a trade-off between fine and coarse granularity required for the implementation of datapath and random logic functions, respectively. This leads to a factor of four improvement in the LUT memory size compared to commercial FPGAs, and, assuming a standard-cell implementation, a 1.6-2.8 lower datapath mapping cost. A modified mixed-grain architecture with the ALU-like functionality reduces the LUT memory size by a factor of 16 compared to commercial FPGAs, and mapped onto standard cells has a 1.9-3.3 times higher datapath mapping efficiency. For these reasons, the proposed FPGA architectures may be an interesting alternative to the traditional general-purpose FPGA devices, especially if characteristics of a target application domain are known a priori.

Categories and Subject Descriptors

B.6.1 [Logic Design]: Design Styles–FPGAs; B.2.4 [Arithmetic and Logic Structures]: High-Speed Arithmetic– Cost/performance; C.3 [Special-Purpose and Application-Based Systems]–Signal Processing Systems

General Terms

Design, Performance

FPGA'03, February 23–25, 2003, Monterey, California, USA.

Keywords

FPGA, DSP, application-domain tuning, datapath optimization, logic block architecture, full-adder, ALU, inverting property, symmetry, routing architecture

1. INTRODUCTION

The aggressive scaling in process technologies and dramatically increasing design and fabrication costs have made FPGAs a cost-efficient alternative for both ASICs and general purpose processors. At the same time, their application area has been expanded from an original 'glue-logic' niche to the system-central position. This has been enabled by the introduction of the so-called platform-FPGAs which integrate a traditional programmable logic fabric with specialized IP cores, and allow an implementation of complete systems on one FPGA chip [24]. The shift towards system-like FPGAs is strongly advocated by main FPGA vendors. They would like to see FPGAs as an ASIC replacement technology in the near future.

In parallel to that, there is a growing interest in using FPGAs in the context of traditional, CMOS-based, systems-on-a-chip. Embedded reconfigurable logic may help there to reduce NRE costs, extend the application life-time, and shorten time-to-market.

What makes FPGAs so attractive is the flexibility they offer. However, the same flexibility, or precisely the way how it is obtained (e.g. look-up table based logic, huge interconnect network), results in a large cost-efficiency gap between FPGAs and ASICs. Finding a good balance between flexibility and efficiency (in terms of area, performance and power) in reconfigurable logic devices is not trivial. This can be seen in the fact that reconfigurable logic architectures have hardly changed over the years. The fabric of today's FPGAs, although enhanced with some extra features, still very much resembles the one used in the first FPGAs; paradoxically, interconnect is made even richer. Therefore, in spite of mature 'know-how' and technological advances which only decrease the absolute cost of a single transistor, FPGA designs still suffer from their high intrinsic cost. This cost, although accepted for some applications, can be a crucial limiting factor for the others.

We address this issue by exploiting the idea of application-domain tuning. This is based on the observation that most FPGAs, although made *general-purpose*, are often used for specific classes (domains) of applications only. The identification of characteristic properties of designs from a given application class allows translating them into architectural improvements. A novel 'mixed-grain' FPGA and its ALU-like modification are presented as examples of such an approach. The architectures of both devices have been optimized for a single application domain only, i.e. DSP, and derived by the analysis of a representative set of DSP benchmarks. We

show that the proposed architectures are superior to any commercial FPGA device if the DSP-type of functions are mapped.

The paper is organized as follows. Section 2 formulates the DSP-tuning problem, while Section 3 surveys existing solutions. In Section 4, the method of application domain characterization is described and applied to DSP. Section 5 describes various properties of arithmetic functions and proposes a structure of a basic logic element which is meant for an efficient implementation of a 1-bit addition. A cost-efficient mixed-grain FPGA logic block which makes use of such a logic element, and its slight modification enabling a further cost reduction, are presented in Section 6 and Section 7, respectively. The next section discusses some interconnect-related aspects, while in Section 9 a comparison method and final results are presented. The conclusions follow in Section 10.

2. PROBLEM DEFINITION

DSP applications are usually considered as being dominated by arithmetic or datapath computations. However, a careful analysis of various DSP benchmarks indicates the presence of other types of computations too, e.g bit-level manipulations or small pieces of random logic. Therefore, for an efficient implementation of DSP, both datapath *and* random logic type of functionality must be supported.

In FPGAs, one of the problems which make this requirement difficult to realize is a different nature of these computations. For example, from the functionality point of view, datapath functions operate on coarser arguments than those which are usually processed by random logic. At the same time, however, the implementation of datapath functions, and in particular arithmetic functions, is usually realized by fine-grain elements, while the implementation of random logic mostly benefits from coarser granularity.

The reason for this 'paradox' is the underlying computing structure of FPGAs, i.e. a LUT (look-up table)-based processing element. The LUT complexity (in terms of the number of its configuration bits) grows much faster with the increase of the bit-width of input arguments for arithmetic than for logic operations [9]. Moreover, the generation of consecutive output bits of arithmetic operations depends on the propagation of a carry signal. If this carry-dependence is coded within the LUT, its size grows exponentially. Since a configuration memory is very costly (especially for embedded systems), the implementation of arithmetic operations in finer LUTs, and a serial carry propagation between them, is preferred [14]. On the other hand, because logic functions are usually implemented as multi-level nets of gates, and have relatively few inputs and outputs, such functions clearly benefit from the implementation in coarser (larger) LUTs. The reason is that such LUTs usually reduce the total logic-depth, and thus the path delay.

In the DSP-optimized FPGAs, the logic block architecture should reflect the conflicting granularity requirements of datapath and random logic functions described above.

3. PREVIOUS WORK

Digital signal processing plays an essential role in many modern applications, and there is a strong need for DSP-specific tuning to meet requirements of such applications. This trend can be observed in the number of publications describing different DSP-optimized FPGA architectures. Based on the chosen optimization technique, several classes of such architectures can be identified. These are:

- *Architectures with dedicated DSP logic.* The DSP-tuning is achieved by the use of hard-wired logic which implements datapath functions. Dependent on the amount of the dedicated logic resources and their overall organization, FPGA architectures of this kind have either a homogeneous structure (all logic blocks are DSP-optimized) [1] or are heterogeneous (an FPGA is divided into regions with DSP-optimized and general-purpose cells) [6]. The third group includes hybrid FPGA architectures which are globally homogeneous and locally heterogeneous (an FPGA has either a hierarchical structure with a heterogeneous lowest level [12], or each FPGA cell has mixed-type components [18]).
- *Architectures of coarse granularity.* In contrast to the conventional bit-level processing, some FPGAs have been made to operate on wider (multi-bit) arguments. Because of the cost-efficiency tradeoff it offers [23], a 4-bit processing has been particularly popular [16][6][15]. Also, in many commercial general-purpose FPGA devices very coarse logic blocks have been used to allow the generation of multi-bit results in one processing element, and, at the same time, the reduction of the routing resource complexity [27][3].
- *Architectures with DSP-specific improvements.* In some FPGA architectures, only small adjustments to the conventional structure have been made to efficiently support DSP functionality. One of the most popular adjustments of this type is the use of dedicated carry logic [25]. The carry logic is faster than the carry signal generated in a LUT. Another technique is sharing of the logic block inputs which allows a multi-bit output to be produced from one set of inputs only (common in datapaths). This technique has been applied both in traditional LUTs [4] and in multi-bit output LUTs [15][13]. Another adjustment technique exploits the fact that slices of datapaths usually implement the same functionality. Thus, configuration bit sharing can be applied to reduce unnecessary area overhead. This technique has been proposed in [6].

4. APPLICATION-DOMAIN TUNING

In conventional general-purpose FPGAs performance is traded for flexibility. This allows to postpone a decision on what precisely is to be mapped, of what complexity, and in what way to the very last moment of the design process. Moreover, it allows to reuse the same piece of silicon for designs of completely different natures.

Though attractive, such flexibility is not always required, and (because of the associated cost) sometimes even undesirable. Therefore, if a given application domain offers optimization opportunities, they should be exploited.

The idea of the application-domain tuning we propose here allows to trade back the ultimate flexibility of FPGA devices for some reduction in their intrinsic cost. This is possible if there exists some pre-knowledge on the type of functions which are to be implemented. A general characterization of such functions and identification of characteristic properties of a given application domain (class) allow to optimize FPGA fabrics. After such an optimization, an FPGA fabric is no longer general-purpose but domain-specific.

In contrast to [7], architectures based on our application-domain tuning concept offer much more flexibility than custom reconfigurable logic which is tailored to a limited set of functions only. On the one hand, this increases the performance penalty, but, on the other hand, it lowers the risk (the mapped functions can still be exchanged with new ones from the same application domain). The implementation of functions with completely different characteristics is possible too, but at a higher implementation cost.

4.1 DSP application-domain characterization

In this section, we present an application-domain analysis performed for the DSP application domain. For this purpose, a rep-

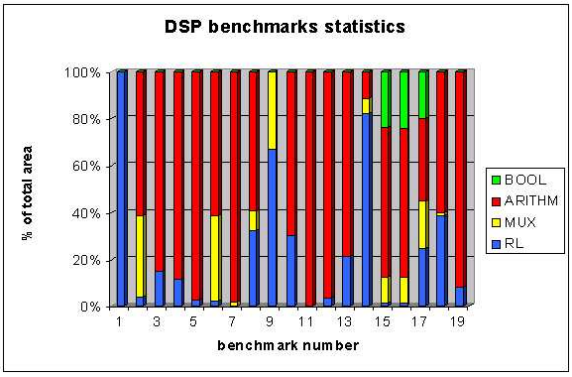

Figure 1: Statistics for 19 industrial DSP designs showing the type and amount of computations required in each design.

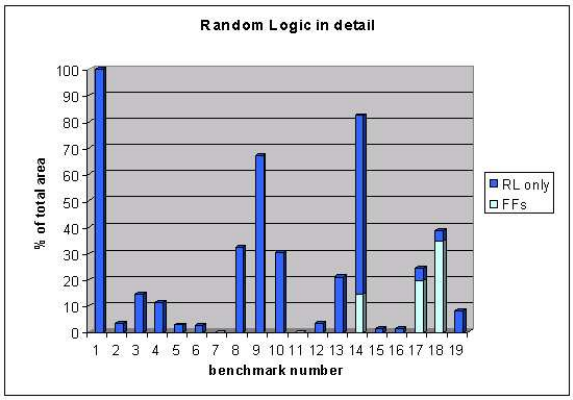

Figure 2: Combinatorial (pure RL) versus sequential (FFs) random logic in the mapped designs.

resentative set of DSP benchmarks is characterized in terms of the type of used computations and the frequency of their occurrence.

We synthesized 19 industrial DSP designs of different complexity: from small ASUs (Application Specific Units) to large filters. We used the Cadence BuildGates synthesis tool-set with AmbitWare datapath library. The latter allows extraction of various macro-blocks, and their preservation (if required) during the mapping process. In our experiments, we set global synthesis parameters such that all datapath components with operands equal or larger than 4-bits and all multiplexers were treated as macro-blocks.

Because a technology independent gate-level netlist generated in this way does not contain information on the area occupied by different components, we mapped our designs onto standard cells from a CMOS 0.13 μm library. For each design, we generated a resource report with the area information.

We classified all design components in two main groups, i.e. datapath logic (implemented by macro-blocks) and random logic (implemented by simple gates). For an accurate analysis of datapath functionality, we distinguished between arithmetic components, multiplexers and wide boolean functions.

In Fig. 1, the information on the properties of the mapped designs is presented in the form of a chart. The chart shows the percentage of area taken by random logic and datapath components. Fig. 2 shows a random logic portion of the designs, and identifies the amount of sequential logic (flip-flops) which is present there. This is to make sure that the identified random logic is not confused with flip-flops.

The obtained results allow to draw the following conclusions:

- As expected, the datapath functionality, and in particular arithmetic, is dominant in DSP. The datapath functions have different bit-widths.
- DSP designs heavily use multiplexers of various size. Thus, an efficient mapping of multiplexers should be supported.
- DSP functions do contain random logic. The amount of random logic varies per design, but on average does not exceed 25% of the total design area. Random logic is predominantly combinatorial.
- Some DSP designs use wide boolean functions. This fact should be reflected in the logic block architecture.

5. COST-EFFICIENT LUT-BASED ARITHMETIC

In this section, we use conclusions of the application-domain analysis presented in Section 4.1 to derive the logic block architecture of a DSP-optimized FPGA device. After discussion on basic assumptions on the logic block architecture, we look closer at different properties of arithmetic functions which can be exploited to reduce the logic block implementation cost.

5.1 General observations

The DSP functionality is dominated by arithmetic. Thus, a cost-efficient mapping requires that fundamental arithmetic operations, such as a binary addition for example, are implemented well.

Though placing a dedicated (hard-wired) adder in an FPGA logic block seems to be the most straightforward way of datapath-tuning, we deliberately give up this scenario. The reason is that, as shown in Section 4.1, in DSP designs a small amount of random logic is also required. Since this amount differs per design, a proper ratio between arithmetic and random logic components in an FPGA architecture is difficult to determine a priori. This questions thus the use of a heterogeneous type of FPGA structures. At the same type, hybrid structures, such as [12], are less attractive since they constrain the mapping tools.

For these reasons, we choose for our FPGA a homogeneous structure and a LUT-based logic block. In this way, both datapath and logic functions can be implemented equally well, and their mapping (in particular placement) does not have to be constrained by a physical location of hardware resources.

5.2 Adder inverting property

The output signals of a full adder, i.e. a sum S and a carry output c_{out}, can be defined as:

$$S = a \oplus b \oplus c_{in} \tag{1}$$

and

$$c_{out} = \bar{c}_{in} \cdot (ab) + c_{in} \cdot (a+b), \tag{2}$$

where a and b are primary adder inputs, and c_{in} is a carry input.

Eqn. 1 and Eqn. 2 can be rewritten as follows.

$$S = \bar{c}_{in} \cdot (a \oplus b) + c_{in} \cdot (\overline{a \oplus b}) \tag{3}$$

$$= \bar{c}_{in} \cdot (a \oplus b) + c_{in} \cdot (\bar{a} \oplus \bar{b}) \tag{4}$$

$$c_{out} = \bar{c}_{in} \cdot (ab) + c_{in} \cdot (\overline{\bar{a} \cdot \bar{b}}) \tag{5}$$

Eqn. 4 and Eqn. 5 represent what is known in the literature as *adder inverting property* [20]. The property describes that the inversion of adder inputs results in the inversion of its outputs.

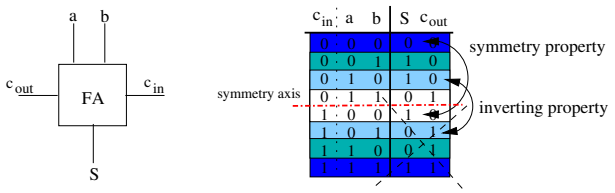

Figure 3: Properties of a binary adder identified in its truth table.

The added inverting property has a nice application to the LUT-based arithmetic as it allows to *halve* the total number of bits required to implement a LUT-based addition. This is possible because of the symmetry in the adder truth table: output bits are the same but of an opposite polarization (see Fig. 3). Such application of the adder inverting property has been proposed in [14], and applied to the implementation of a cost-efficient multi-functional logic block with 2-bit or 4-bit granularity [15]. The logic block of this type uses a multi-bit output LUT implemented in a memory-like way, and can be configured to work in a datapath-, logic- or memory-mode [15].

5.3 Symmetry property

Next to the inverting property, the *symmetry* of the adder sum function S, as described by Eqn. 3, is of an interest too. Symmetrical functions do not change by permuting their inputs [5]. In practice, for the LUT-based arithmetic it means, for example, that the adder sum function can be generated, dependent on the polarization of a reference signal, either by a direct calling of the base function F stored in a LUT, or by its inversion $\bar{F}$. If we choose $c_{in} = 0$ as the reference signal, and implement in the LUT the base function $F = a \oplus b$, then

$$S = c_{in} \cdot F + \bar{c}_{in} \cdot \bar{F} \tag{6}$$

5.4 Optimal LUT-based adder implementation

Though both the adder inverting property and the symmetry property offer the same optimization gain: a factor of two reduction in the number of LUT memory bits, the symmetry property is a superior technique. This is because the area and performance penalty of implementing this technique by means of a conventional LUT is lower than the same penalty for the inverting property.

A potential disadvantage of the symmetry property is its applicability to the adder sum bit only. This can easily be overcome if dedicated logic is used for the carry signal implementation. Such an approach improves the overall timing (a carry signal is usually on the critical path), and the logic utilization (no LUTs have to be sacrificed to implement the carry).

Fig. 4 shows an optimal structure of a 1-bit LUT-based addition. The structure makes use of the symmetry property described by Eqn. 6. This is realized by an XOR gate at the LUT output. The gate implements a programmable inversion. The LUT is a 2-input LUT and stores the adder sum output bits coded for one polarization of the carry input signal (e.g. $c_{in} = 0$). The analogous outputs for the carry signal of an opposite polarization (i.e. $c_{in} = 1$) are generated by the inversion of the LUT contents.

The carry signal is implemented in a way suggested in [28], this is with an XOR gate and a 2:1 multiplexer. However, because in the arithmetic mode each 2-LUT is configured as an XOR gate, it is possible to reuse the LUT signal in the carry signal generation. This is similar to the carry generation technique used in Xilinx Virtex

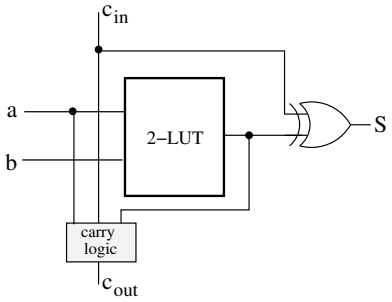

Figure 4: An optimal 1-bit LUT-based adder implementation.

devices [26]. In this way, not only few extra gates are saved, but the carry chain can be break in an arbitrary place too. This is essential for implementing arithmetic functions with less than four bits. The chosen carry implementation method has no negative influence on timing.

We call the above described structure *optimal* since it requires the minimum number of configuration bits, i.e. four, to implement a 1-bit LUT-based addition. This is a factor of *four* improvement compared to any state-of-the-art FPGA device. In FPGA device families such as Xilinx Virtex [26][27] and Altera FLEX [2], a 4-LUT with 16 memory cells is used to implement a sum output bit of a 1-bit addition; in ATMEL AT40K devices [4], Altera Stratix family [3] or Low-Power FPGA from Berkeley University [9], 16 configuration bits in total are used to implement a 1-bit addition, i.e. both sum and carry output signals.

Such an aggressive reduction in the number of configuration bits decreases not only the cost of implementing arithmetic functions, but also the total size of the configuration memory. The latter is important especially for FPGA devices supporting partial or dynamic reconfiguration, or any embedded FPGA (due to an on-chip memory).

6. MIXED-GRAIN FPGA LOGIC BLOCK

In this section, a novel FPGA logic block which allows an efficient implementation of datapath functions, and having capabilities of mapping random logic functions too, is presented. The logic block uses a 2-LUT with symmetry property introduced in Section 5.4 as a basic logic element.

6.1 Basic concept

In Section 2, we have described the problem of tuning of an FPGA logic block to the DSP application domain as being determined by a different nature of computations in datapath and random logic functions. The key feature of the solution we propose here is the preservation of this nature, and expressing it in the way how these different computations are implemented, without sacrificing the overall efficiency.

What we observe is that datapath functions produce a multi-bit output, and require relatively simple logic elements[1] for their implementation. Also, datapaths are very regular and have a bit-sliced structure. At the same time, random logic functions usually produce a single-bit output and benefit from more complex (coarser) logic elements. With such a characteristic of datapath and random logic functions, the problem of the DSP-specific optimization of an FPGA logic block is how to properly make use of its LUT configuration bits.

[1]We assume here that the complexity of a logic element is a function of its configuration memory size.

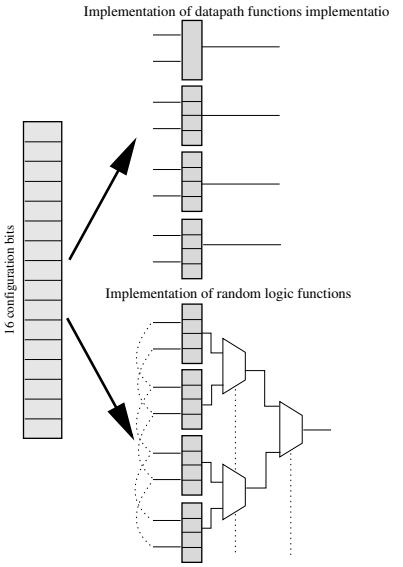

Figure 5: Reuse of the LUT configuration bits to support implementation of different type of computations.

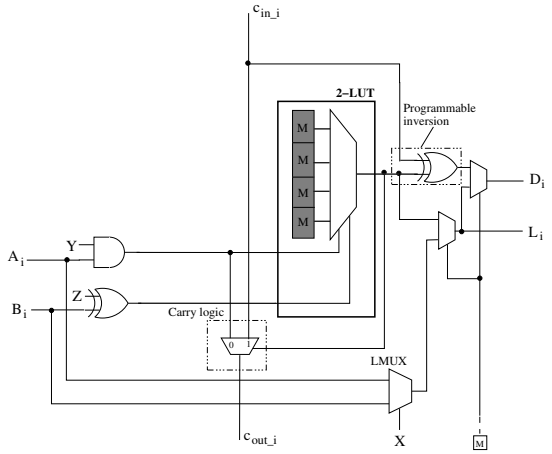

Figure 6: Basic logic element of the novel logic block, which implements a bit slice of datapath functions.

Assuming that a 4-LUT is a good candidate for random logic mapping [21], and exploiting the fact that a 2-LUT is sufficient to implement basic arithmetic functions (see Section 5.4), we determine the optimal complexity of our logic block as having 16 configuration bits. The basic idea we exploit here is that these 16 configuration bits can be treated as either one element (a 4-LUT) if random logic functions are to be mapped, or they can be decomposed into four groups of 4 configuration bits (four 2-LUTs) if datapath functions are to be mapped. This is shown in Fig. 5. The logic block implemented in this way allows to implement 4-bit datapath functions (i.e. operating on 4-bit arguments and producing a 4-bit result), and random logic functions (with up to four inputs and a single-bit output). To reflect the granularity trade-off such a processing element offers, we call it a *mixed-grain logic block*.

6.2 Bit-slice structure

A basic logic element of the proposed logic block implements a bit-slice of datapath functions. The logic element is based on the structure introduced in Section 5.4, and consists of a 2-LUT, an XOR gate and a dedicated carry logic circuitry.

To allow an implementation of arithmetic functions other than a 1-bit addition for which this structure has been optimized, two extra gates, i.e. an XOR gate and AND gate, are placed at the LUT inputs. The AND gate is used in a multiplier-mode and allows an array multiplier cell to be implemented in a single bit-slice of the logic block. The gate implements a logical AND of the data input A_i and the global signal Y. The XOR gate at the second LUT input is used for the implementation of the addition or subtraction operations, which can be chosen dynamically by setting the global signal Z ($Z = \overline{ADD}/SUB$). The signal Z and the data input B_i are inputs of the XOR gate.

The results of the application-domain characterization discussed in Section 4 reveal the presence of a considerable number of multiplexers in DSP designs. The importance of having support for the efficient implementation of multiplexers in an FPGA logic block has also been confirmed by Agarwala in [1] and Cherepacha in [6]. Therefore, we assume that a bit-slice must allow mapping of a 2:1 multiplexer. However, the multiplexer function ($F = a \cdot \bar{c} + b \cdot c$, where a and b are multiplexer inputs and c is a selection signal) is not symmetrical, and its mapping onto the proposed logic element is not directly possible. Since placing of a dedicated 2:1 multiplexer in our logic element is better from both area and performance point of view, we choose this option[2].

The 2:1 logic multiplexer (LMUX in Fig. 6) is placed in parallel to the 2-LUT. Its inputs are two data inputs of the logic element (i.e. A_i and B_i), and a selection signal is the global signal X.

The structure of the logic element is shown in Fig. 6. The element has two outputs: a datapath output D_i and a random logic output L_i. The D_i output is used when the logic element is configured as a datapath bit-slice and a *multi-bit* result is generated, while the L_i output is used in the random logic mode when a *single-bit* result is produced. The LMUX, 2-LUT and XOR gate outputs are multiplexed to allow an implementation of wide boolean functions or wide multiplexers (both generate multi-bit output).

6.3 Logic block structure

The novel logic block consists of four logic elements (slices), with a structure as shown in Fig. 6. The choice of 4-bits as a granularity of the logic block is dictated by the fact that such granularity has been found the most optimal for the datapath mapping [6][23].

The logic outputs of each slice are merged together in global multiplexers MUX1, MUX2 and MUX3, while datapath outputs are fed to the output selection stage (see Fig. 7). Additionally, the outputs of multiplexers MUX1 and MUX2 are fed to the output selection stage, where they are multiplexed with datapath outputs of the first and third bit-slice, respectively. These outputs are used if a 2-bit 4:1 multiplexer is implemented.

The logic block has eight primary inputs $IN_0 \ldots IN_7$, three control inputs C_0, C_1, C_2, four primary outputs $OUT_0 \ldots OUT_3$, and a special carry output C_{OUT}. The control signals are multiplexed with the static signals '0' and '1' defined by the configuration bits, and produce a set of signals X, Y, Z, respectively. The X, Y, Z signals are global for all bit-slice. The sharing of these signals is possible because of the datapath regularities. As a result, this allows to reduce the number of logic block pins, and consequently the routing resource complexity. The carry input signal C_{IN} on pin C_0 is fed to the first bit slice only. This slice produces an intermediate carry output signal c_{out_0} which is connected with the carry input of the

[2]There is a sort of 'partial' symmetry in a multiplexer function, but its realization is rather costly.

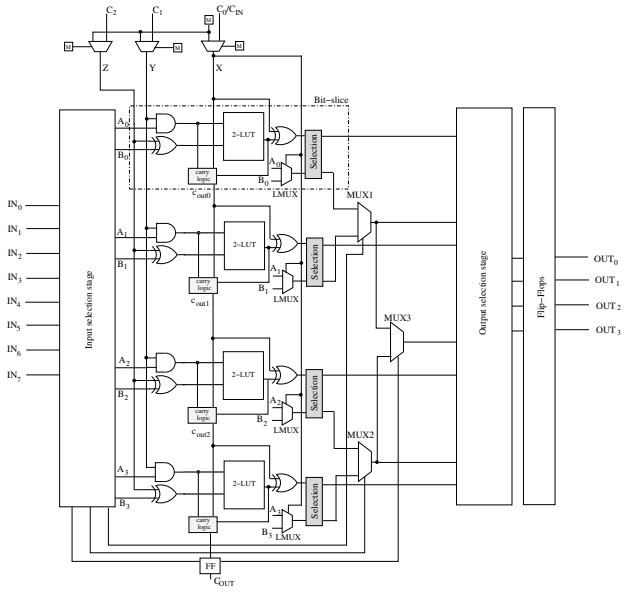

Figure 7: The mixed-grain logic block architecture.

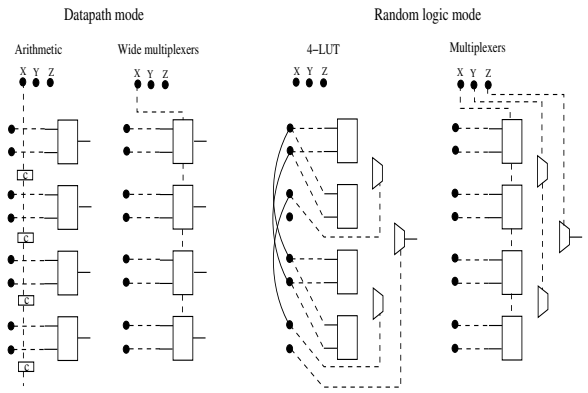

Figure 8: The connectivity between inputs of the logic block and inputs of the consecutive bit-slices.

next (second) bit-slice, and so on.

The task of the *input selection stage* is to define the connectivity between inputs of the logic block and inputs of each logic element (LUT) dependent on the required functional mode. A similar function the *output selection stage* has. Each of the logic block outputs can be registered by means of dedicated flip-flops.

6.4 Functional modes

The mixed-grain logic block has two primary functional modes:

- *Data-path mode* in which a multi-bit output (max. 4-bit) is produced using D_i outputs of each logic element; all bit-slices are configured to implement the same function.
- *Random logic mode* in which a single-bit output is produced using L_i outputs of each logic element; bit-slices usually implement different logic functions.

We assume that only one of these modes is possible at the same time. If the number of bit-slices required to implement a given function is smaller than the total number of the logic block slices (i.e. four), the remaining logic block resources are left unused. Although this might slightly influence the total number of logic blocks required to implement such functionality, it reduces the complexity of the control structure in each logic block. Of course, a more flexible implementation is possible too. Below, the logic block functionality in each of the modes is discussed in detail. Fig. 8 shows connectivity examples in each mode.

6.4.1 Datapath mode

Addition/Subtraction. The logic block can be configured to implement up to 4-bit addition/subtraction operations. The type of an operation can be determined statically or dynamically by connecting the signal $Z = \overline{ADD}/SUB$ to a local or global signal, respectively. A 2-LUT in each bit-slice stores half (see discussion in Section 5.4) a truth table of a 1-bit adder. If a binary addition is implemented, the input XOR gate passes the input operand B_i without changing its polarization, i.e. Z='0'. If a subtraction is implemented, Z='1' and the XOR gate negates one of the operands. The input AND gates are unused, i.e. signal Y='1'. The dedicated carry logic implements the carry path. For operations with less than four bits, unused slices have their LUTs programmed to generate the value '1' such that the carry logic allows the carry signal from the previous slices to be propagated to the output (C_{OUT}). The outputs (sum/decrement bits) are available on the datapath outputs of each logic element.

Multiplication. The logic block supports an implementation of an array multiplier with a ripple carry adder [20]. Maximally, a 4-bit section of such a multiplier (i.e. four cells) can be implemented in one logic block. For this purpose, each bit-slice is configured as a binary adder (inactive XOR gates, i.e. Z='0') with an AND gate on one of its inputs. The signal Y carries the value of the multiplicand bit. In this mode, four partial product signals are generated on the datapath outputs, and the carry output signal is available on the C_{OUT} pin.

Wide multiplexers. In the datapath mode, the multiplexers in the logic block have multi-bit inputs and outputs. Maximally, a 4-bit 2:1 multiplexer or a 2-bit 4:1 multiplexer can be implemented in one logic block. In the first case, LMUXes of each bit-slice are used, and the result is available on the datapath outputs. In the second case, each slice produces output bits on the logic outputs of each bit slice, and the global multiplexers MUX1 and MUX2 are used to generate two final signals. The result is available on the first and third output pins of the logic block. A wide 2:1 multiplexer uses global signal X as a control signal, and a wide 4:1 multiplexer uses signals X and Y as control signals.

Wide boolean functions. The logic block allows the implementation of boolean functions of two multi-bit operands. Such functions are implemented by storing the same truth table of a given boolean function in each 2-LUT. In each bit-slice, the LUT output is selected in an upper selection multiplexer, and connected to the datapath output.

6.4.2 Random logic mode

Boolean functions. Logic (boolean) functions of up to four inputs can be implemented in the logic block. A 4-input function is decomposed using Shannon expansion theory [17], and mapped onto a set of four 2-LUTs and global multiplexers MUX1, MUX2 and MUX3, which merge the LUT outputs. Each bit-slice produces a result available on the logic output. The 3-input/2-output logic functions can also be realized in a single logic block.

Multiplexers. The mapping of multiplexers with a single output and up to eight inputs is possible. If the largest multiplexer, i.e. a 1-bit 8:1 MUX, is to be implemented in the logic block, all global multiplexers MUX1, MUX2 and MUX3 are used. In this case,

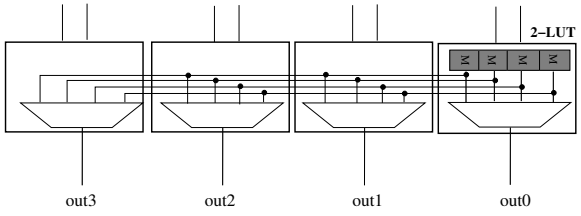

Figure 9: Configuration bit sharing in LUTs of the modified ALU-like logic block.

LMUXes in each slice and multiplexers MUX1-MUX2 and MUX3 are controlled by global X, Y, and Z signals, respectively. The data inputs of the multiplexers are inputs of the logic block, and the final output is available on all logic block outputs.

7. MODIFIED LOGIC BLOCK WITH ALU-LIKE FUNCTIONALITY

The key feature of the logic block structure proposed in Section 6 is its ability to implement efficiently both datapath and random logic functions. However, in some applications the random logic functionality is not required, and what is more it unnecessarily increases the mapping cost. At the same time, bit-slices of datapath functions implement usually the same function, thus LUTs with independent sets of configuration bits offer a redundant functionality.

These observations indicate that our logic block structure can be simplified even further. To realize this, we utilize the concept of the configuration bit sharing [6]. As a result, four independent 2-LUTs can be replaced by one set of four configuration bits and four sets of the decode logic (multiplexers). This is illustrated in Fig. 9.

Assuming that the rest of the logic block structure is left unchanged, such an optimization technique reduces the LUT memory size by a factor of four compared to the proposed mixed-grain logic block architecture, and by a factor of 16 compared to any commercial LUT-based logic block architecture.

An interesting observation is that the logic block structure obtained in this way resembles a traditional 4-bit ALU with LUT configuration bits as instruction bits. The main advantage of our solution is, however, a higher flexibility since *any* logic function of two inputs, various multiplexers, and array multiplier cells can also be implemented.

8. INTERCONNECT ISSUES

Because the cost-efficiency of today's FPGAs is strongly determined by the architecture and implementation of their routing resources, the discussion on new FPGA architectures cannot be limited to the FPGA logic blocks only. Therefore, in this section we discuss some interconnect-related issues. We show that the increased functional capacity of the proposed logic blocks does not have a negative impact on the overall complexity of its routing resources. In our analysis, we assume that the routing resource complexity is a function of the number of logic block pins [10].

8.1 Routing architecture improvements

As discussed in Section 2, the FPGA logic block structure is influenced by the different nature of datapath and random logic computations. Similarly the interconnect structure is. In the implementation of datapath functions, many routing tracks are required to route multi-bit arguments of datapath operations. But, because of the datapath regularity, there is a strong correspondence between the logic block pins and the track assignment: most of the datapath signals are routed like buses, i.e. following a specific order. In

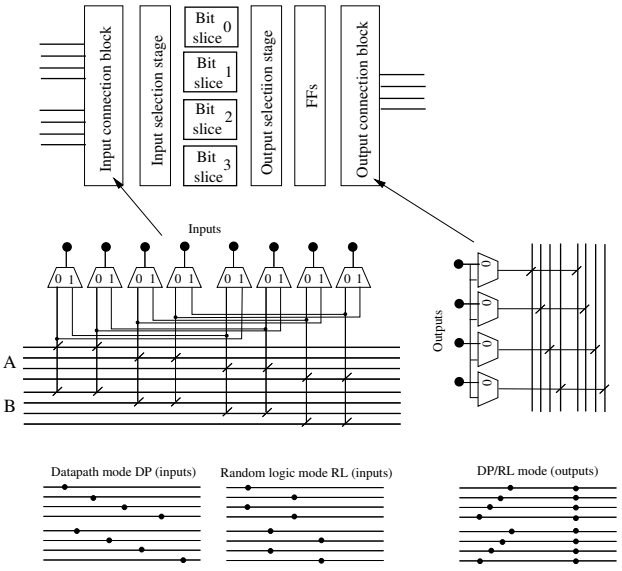

Figure 10: Proposed implementation of the programmable interconnect (example).

contrast to that, random logic functions require relatively few wires because of the limited number of LUT pins. However, the overall irregularity of such functions causes that a much higher degree of flexibility is required in the routing resources. Thus, a very rich interconnect structure is needed.

The fact that the proposed logic block offers a different functional capacity for datapath and random logic functions can be well exploited. Fig. 10 shows a basic structure of our logic block with an example implementation of input and output connection blocks[3]. In the input connection block, an extra layer of multiplexers enables the selection of a routing track to which a LUT pin connects to. We take advantage of a one-to-one correspondence between pins and routing tracks which is typical for a datapath mode (the control signals of multiplexers are set to '0'). Furthermore, in the random logic mode, the additional routing tracks are used to increase the connection flexibility (the control signals of multiplexers are at any value). The is possible because in the random logic mode only four signals instead of eight (as in the datapath mode) are used (a 4-LUT implementation). A similar idea is applied to logic block outputs. In the random logic mode only one valid signal is produced. By placing an additional layer of 2:1 multiplexers, it is possible to select to which of the four output pins the output signal connects to. In this way, the total load on the routing tracks can be decreased since smaller multiplexers are required to implement the input and output connectivity. This influences both performance and power consumption. Furthermore, the nibble datapath operations are implemented locally (i.e. in one logic block), instead of being distributed among different logic blocks (e.g. [4]).

8.2 Complete routing architecture

The definition of a routing structure is necessary to establish the total cost of a given FPGA architecture (the so-called logic tile). Therefore, we briefly describe here a routing structure chosen for the two proposed FPGAs. Although this structure has been found by an analysis of the requirements of both datapath (manual mapping of small DSP kernels) and random logic functions (automatic

[3]The connection blocks define the connectivity between the logic block pins and the associated routing tracks.

Table 1: Area and performance comparison for different FPGA architectures (figures for the proposed architectures refer to their standard-cell instead of a full-custom implementation).

FPGA architecture	CLB organization	A_{tile} [λ^2]	T [ns]	MC_{DP} [λ^2/bit]	MC_{RL} [$\lambda^2/4-LUT$]
Xilinx 5K	4 x 4-LUT	2.24M	6	1.12M	0.56M
ORCA 2C	4 x 4-LUT	4.25M	7	1.1M	1.1M
Xilinx 4K	2 x 4-LUT	1.25M	7	0.63M	0.63M
Altera FLEX8K	8 x 4-LUT	7.4M	7.5	0.93M	0.93M
LEGO	4 4-LUT	4.1M	4.1	2M	1.16M
LP-PGA II	5/2-LUT	3.38M	8	1.69M	2.25M
CFPA	4-bit logic	1M	4	0.25M	0.25M†
CHESS	4-bit ALU	1.4M	5	0.35M	0.35M†
Mixed-grain	4-LUT	1.6M	1.6	0.4M	1.6M
ALU-like	4 x 2-LUT+shared mem	1.3M	1.5	0.33M	0.33M†

mapping of MCNC benchmarks using SIS, FlowMap and VPR), it has a preliminary character only, and a more careful routing analysis will be require in the future.

We have assumed a uniform routing architecture with a channel width W=30. The channel comprises two 4-bit buses of the length one, two 4-bit buses of the length four, and two 4-bit buses of the length eight. Next to the general-purpose routing, each channel also includes six special tracks for routing of the logic block control signals (C_0, C_1 and C_2 on Fig.7). The number of tracks of a given type and the locations of the connection boxes are established taking into account differences in the nature of the data and control flows. To support the mapping of datapath macro-blocks, which heavily rely on the local connections (intra-block communication), we provide a set of direct connections in each of the directions (i.e. horizontal, vertical and diagonal). Each logic block produces one 4-bit direct output and accepts two 4-bit direct inputs.

The connection block is implemented in the way described in Section 8.1. This results in two different connection block flexibilities F_c [22] dependent on the operating mode of the architecture. In the datapath mode, F_{ci}=0.25 and F_{co}=0.25, while in the random logic mode, F_{ci}=0.5 and F_{co}=1 (F_{ci} and F_{co} are connection block flexibilities for input and output pins, respectively). In the mixed-grain logic block, the logic pins connect to the routing tracks independently (i.e. connections are selected by the independent set of configuration bits), while in the ALU-like architecture, the input connection pattern is limited (i.e. configuration bits are shared).

We use a switch box type as in the Atmel AT40K architecture [4]. Such a switch box requires only three instead of six switches. Although this limits the routing flexibility, our mapping experiments using the modified VPR show that such flexibility is sufficient for mapping datapath-dominated designs. In the mixed-grain architecture, switches of two tracks are controlled by the same set of configuration bits (pin equivalence), while in the ALU-like architecture, switches of four tracks are controlled by one set of bits only (bus routing).

A slightly different character of the two proposed architectures influences their total implementation cost. Due to simplifications in logic and routing, the ALU-like architecture requires only 71 configuration bits, as opposed to 152 in the mixed-grain architecture.

9. COMPARISON

In this section, we present results of the comparison of our two FPGA logic block architectures, i.e. the mixed-grain logic block architecture and the modified logic block architecture with the ALU-like functionality, with logic block architectures of different state-of-the-art FPGA devices. Because architectural and implementation details of most of the commercial FPGAs are, to a large extent, proprietary, a truly objective comparison is very difficult. To resolve this issue, we propose two comparison methods that allow to reason about the cost of FPGA architectures in an objective way.

In the first method, we assume that silicon area and performance are primary cost measures, and we use them to characterize state-of-the-art and the two proposed architectures. In this comparison, the state-of-the-art architectures are represented by four commercial architectures and four FPGA architectures which have proposed in the literature. The area and performance figures for commercial architectures come from the similar comparison presented in [8], while the latter are characterized by the figures from original publications where these architecture have been introduced, i.e. [19][9][12] and [16], respectively.

To obtain results of the comparable accuracy also for the proposed architectures, we implement them together with the chosen routing structure (see Section 8). To minimize a design effort, we use a *standard-cell* implementation instead of a full-custom one. Each architecture is designed in the Cadence design environment (schematic entry), and functionally simulated using Pstar. Pstar is also used to derive the timing information. The written-out Verilog netlists are used as inputs for the synthesis tool (Cadence Ambit), and are mapped onto the TSMC CMOS 0.13 μm standard cell library. For the ease of the synthesis process, all programmable switches are replaced by standard cells of a similar size. A relatively large [8] configuration memory cell, i.e. 3kλ^2/bit, is assumed. The final area figures are obtained from Ambit area reports.

Table 1 shows the results. For each architecture, we provide information about its organization, and we list two parameters:

- A_{tile} which is the total logic tile area (i.e. logic block and its routing area) converted into the technology independent λ^2 measure, and
- T which is a cycle time for a logic block and its associated routing in *ns*.

Additionally, for each architecture we calculate two extra parameters which reflect their mapping capabilities. The first parameter, MC_{DP}, describes the mapping cost in silicon area of a 1-bit datapath function (e.g. an adder), while the second, MC_{RL}, is the mapping cost of random logic expressed in silicon area per a 4-LUT equivalent[4]. The positions marked with '†' mean that a given logic block architecture does not implement a full 4-LUT, but (a set of) wide boolean functions only. The values of MC_{DP} and MC_{RL} are shown in λ^2.

Because Table 1 gives information about the commercial FPGA architectures which are slightly obsolete, we propose a second (complementary) comparison method as well. In this method, we analyze very recent and currently widely used commercial FPGAs,

[4]Logic block architectures such as [19] and [9] share some of the LUT inputs. To reflect this fact, we use slightly smaller capacity factors, i.e. 3.5 and 1.5, instead of 4 and 2, respectively.

such as Xilinx Virtex [26], Altera Stratix [3] and Atmel AT40K [4], and compare them with our architectures. As mentioned before, for these architectures area figures are not available and we use an area estimation technique based on the area model suggested in [10]. This model estimates the complexity of an FPGA architecture using two essential parameters: N_{lmb} - the number of the LUT configuration bits, and N_{pins} - the total pin number of a given architecture. N_{lmb} reflects the complexity of the logic block, while N_{pins} the complexity of the routing resources. The latter is based on the assumption that each architecture pin has a fixed routing cost, which is a reasonable assumption. Because of its simplicity and the sufficient level of accuracy, this model (especially for the routing complexity estimation) has been quite popular in the past [10][6][11][22].

Table 2 and Table 3 show the results of the mapping efficiency comparison using the described above model. Eight simple yet representative datapath and logic functions are chosen as benchmarks. For each commercial architecture and our architectures, the tables list three parameters: N_{CLB} which is the number of logic blocks of a given type required to map a benchmark function, and the total mapping cost *MC* which is characterized by two numbers: $Bits = N_{CLB} \cdot N_{lmb}$ and $Pins = N_{CLB} \cdot N_{pins}$, with N_{lmb} and N_{pins} defined as above. The *Bits* and *Pins* are the total number of LUT bits and architecture pins required to map a benchmark.

For each compared architecture, we count only primary inputs and output pins, including carry pins, but excluding registered output pins. Thus, the commercial FPGAs are characterized in the following way: Xilinx Virtex: N_{pins}=28, Altera Stratix: N_{pins}=53, Atmel AT40K: N_{pins}=6. In the two proposed FPGA architectures, a large number of pins is a result of the mulit-bit datapath support. However, since mapping of datapath functions require much less routing flexibility, the routing cost associated with each 'datapath' pin is much lower than with an equivalent 'random-logic' pin. This can also be seen in the fact that in the random logic mode our logic blocks use only five pins, i.e. four inputs and one output (excluding control pin count). To express this difference, we count each 'datapath' pin of our architectures as 0.5 of the 'random-logic' one[5]. With this assumption, our logic blocks have in total N_{pins}=10. The number of configuration bits per CLB for each architecture is as follows: Xilinx Virtex: N_{lmb}=64, Altera Stratix: N_{lmb}=160, Atmel AT40K: N_{lmb}=16, mixed-grain: N_{lmb}=16, ALU-like: N_{lmb}=4.

9.1 Analysis of results

The results from Table 1 show that the two presented FPGA architectures are - regardless their standard-cell-based implementation and no optimizations - superior to the commercial LUT-based FPGAs implemented in a full-custom way. The cost of mapping datapath functions is for the mixed-grain and the ALU-like architectures a factor of 1.6-2.8 and a factor of 1.9-3.3, respectively, lower than a similar cost calculated for the commercial FPGAs. The main reasons for this is an efficient use of the LUT configuration bits (based on the symmetry property described in Section 5.3), and an efficient routing architecture in which the connection flexibility offered by extra 'datapath' pins is fully exploited.

Although optimized primarily for datapaths, the mixed-grain logic block architecture allows an efficient mapping of random logic circuits too. The mapping cost in this case shows that our architecture is only 2.9 times worse than the best in this category Xilinx 5K.

If the random logic functionality is less important, the proposed ALU-like logic block structure is a good alternative. This architecture is 19% smaller, and requires less then 50% of the total number of configuration bits of the mixed-grain architecture. The area advantage comes mainly from the simplification of the routing structure, and not from the reduction of the LUT memory size.

Our architectures compare well also with arithmetic-optimized FPGA architectures proposed in the literature. Because of the similar functionality, especially the CFPA and CHESS architectures are interesting. The mapping cost of the ALU-like architecture is 32% higher than for CFPA, and 6% lower than for CHESS. The datapath mapping cost of the mixed-grain architecture is 60% and 14% higher, respectively. The random logic mapping cost is also much higher. The reason of the area overhead in the novel architectures is their higher flexibility (especially true for the mixed-grain architecture), and their standard-cell implementation.

Tables 2 and Table 3 show the mapping efficiency comparison between the novel architectures and recent commercial FPGA architectures. Our architectures show a factor of 4 and 16, respectively, reduction in the number of LUT configuration bits if datapath functions are mapped. This is independent on which commercial FPGA is compared. However, since the influence of routing is much more severe, the overall benefit of the novel architectures is lower. The comparison of the logic pins number (*Pins*) which reflects the routing complexity shows only a factor of 2.2-3 improvement. This is in line with the improvement factors calculated from Table 1.

10. CONCLUSIONS

Today's FPGAs are a cost-efficient alternative for both ASICs and programmable processors. Nevertheless, their general-purpose nature makes them much more costly and much less efficient than standard-cell-based or custom designs. To address this issue, we proposed a way of the intrinsic cost reduction for FPGAs by tuning their architectures to an application domain or a class of applications. We described a method of the application-domain characterization which allows to identify an essential type of functionality that has to be mapped efficiently. We applied this method to characterize DSP applications, and we found that arithmetic components, multiplexers, wide boolean functions, but also a small amount of random logic are required in DSP.

Based on these observations, we proposed a 'mixed-grain' FPGA logic block architecture which exploits characteristic properties of arithmetic functions. The novel logic block offers a trade-off between coarse and fine granularity, and can be used to map efficiently both multi-bit output datapaths and single-bit output random logic functions. The logic block architecture is such that it can directly be used in conventional FPGAs. The modified mixed-grain logic block which reduces the implementation cost even further has also been presented. The modified architecture has the ALU-like functionality and very few configuration bits.

We implemented the novel architectures using standard cells, and compared them with fully optimized commercial FPGAs. For such a comparison, the area improvement factor calculated per bit of the datapath functionality is between 1.6 and 3.3. The LUT memory size is reduced by a factor of 4 and 16, respectively, compared to the LUT-based FPGA devices. Both our architectures are very well suited for datapath mapping, and the mixed-grain architecture shows only a small overhead when random logic functions are mapped. The overall cost-efficiency of the described FPGA architectures positions them between general-purpose LUT-based FPGAs and traditional ASICs. The mixed-grain architecture, although of a slightly higher cost than the datapath-optimized architectures known from the literature, offers much higher flexibility. A full-custom implementation of new architectures would increase their cost-efficiency even further.

[5] Similar method to estimate routing has been used in [6].

Table 2: Mapping efficiency of the commercial LUT-based FPGA architectures.

Benchmark function	Xilinx Virtex			Altera Stratix			Atmel AT40K		
	N_{LB}	MC		N_{LB}	MC		N_{LB}	MC	
	–	*Bits*	*Pins*	–	*Bits*	*Pins*	–	*Bits*	*Pins*
8-bit ADD	2	128	56	0.8	128	43	8	128	48
16×16 MULT	68	4.25K	1.9k	73	11.4K	3.9k	256	4K	1.5k
2:1 MUX/4b	1	64	28	0.4	64	22	4	64	24
8:1 MUX/1b	1	64	28	0.5	80	27	7	112	42
16:1 MUX/1b	2.5	160	70	1	160	53	15	240	90
2-inp OR/4b	1	64	28	0.4	64	22	4	64	24
3-inp OR/1b	0.25	16	7	0.1	16	6	1	16	6
4:16 DECOD	4	256	112	1.6	256	85	16	256	96

Table 3: Mapping efficiency of the proposed FPGA architectures.

Benchmark function	**Mixed-grain**			**ALU-like**		
	N_{LB}	MC		N_{LB}	MC	
	–	*Bits*	*Pins*	–	*Bits*	*Pins*
8-bit ADD	2	32	20	2	8	20
16×16 MULT	64	1K	640	64	258	640
2:1 MUX/4b	1	16	10	1	4	10
8:1 MUX/1b	1	16	10	1	4	10
16:1 MUX/1b	3	48	30	3	12	30
2-inp OR/4b	1	16	10	1	4	10
3-inp OR/1b	1	16	10	2	8	20
4:16 DECOD	12	192	120	12	48	120

Because most of today's applications use signal processing algorithms which heavily rely on arithmetic computations, the proposed here FPGA architectures can be interesting alternatives to commercial FPGAs which are often used for this purpose. Furthermore, a considerable reduction of the configuration memory size makes them very attractive for embedded systems and for devices with partial or dynamic reconfiguration.

11. REFERENCES

[1] M. Agarwala and P. Balsara. An architecture for a DSP Field-Programmable Gate Array. *IEEE Transactions on VLSI Systems*, 3(1):136–141, March 1995.

[2] Altera. *FLEX 10KE Programmable Logic Device Family. Data sheet*. Altera, 2000.

[3] Altera. *Stratix Programmable Logic Device Family. Data sheet*. Altera, 2002.

[4] Atmel. *5K-50K Gate FPGA with DSP Optimized Core Cell and Distributed FreeRAM. Summary*. Atmel, 1999.

[5] N. F. Benschop. Symmetric logic synthesis with phase assignment. In *Proc. of the 22nd Symposium on Information and Communication Theory*, pp. 115–122. WIC, March 2001.

[6] D. Cherepacha and D. Lewis. DP-FPGA: An FPGA architecture optimized for datapaths. *VLSI Design*, 4(4):329–343, 1996.

[7] K. Compton and S. Hauck. Totem: Custom reconfigurable array generation. In *Proc. of IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, April 2001.

[8] A. DeHon. *Reconfigurable Architectures for General-Purpose Computing, AI Technical Report 1586, MIT Artificial Intelligence Laboratory*. MIT, 545 Technology Sq., Cambridge, MA 02139, 1996.

[9] V. George. *Low Energy Field-Programmable Gate Array, Ph.D. Thesis*. University of California, Berkeley, 2000.

[10] J. He and J. Rose. Advantages of heterogeneous logic block architectures for FPGAs. In *Proc. of IEEE Custom Integrated Circuits Conference*. IEEE, May 1993.

[11] D. Hill and N.-S. Woo. The benefits of flexibility in look-up table-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):349–353, February 1993.

[12] A. Kaviani, D. Vranesic, and S. Brown. Computational Field Programmable Architecture. In *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 261–264. IEEE, May 1999.

[13] K. Leijten-Nowak, A. Katoch. Architecture and implementation of an embedded reconfigurable logic core in CMOS 0.13μm. In *Proc. of 15th IEEE ASIC/SOC Conference*. IEEE, September 2002.

[14] K. Leijten-Nowak and J. L. van Meerbergen. Applying the adder inverting property in the design of cost-efficient reconfigurable logic. In *Proc. of 44th IEEE Midwest Symposium on Circuits and Systems*. IEEE, August 2001.

[15] K. Leijten-Nowak and J. L. van Meerbergen. Embedded reconfigurable logic core for DSP applications. In *Proc. of Field-Programmable Logic and Applications Conference*, pp. 89–101, September 2002.

[16] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, B. Hutchings. A Reconfigurable Arithmetic Array for multimedia applications. In *Proc. of ACM Symposium on FPGAs*, February 1999.

[17] G. D. Michelli. *Synthesis and Optmization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[18] N. Miller and S. Quigley. A novel Field Programmable Gate Array architecture for high speed processing. In *Proc. of Field-Programmable Logic and Applications Conference*, pp. 386–390, September 1997.

[19] J. R. K. C. G. P.-M. Paul Chow, Soon Ong Seo and I. Raharadja. The design of an SRAM-based Field-Programmable Gate Array. Part II: Circuit design and layout. *IEEE Transactions on VLSI Systems*, 7(3):101–110, September 1999.

[20] J. Rabaey. *Digital Integrated Circuits. A Design Perspective*. Prentice Hall, 1996.

[21] J. Rose, R. Francis, P. Chow, and D. Lewis. The effect of logic block complexity on area of programmable gate arrays. In *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 5.3.1–5.3.5, May 1989.

[22] J. R. Z. V. S.D. Brown, R.J. Francis. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[23] T. Stansfield. Wordlength as an architectural parameter for reconfigurable computing devices. In *Proc. of Field-Programmable Logic and Applications Conference*, pp. 667–676, September 2002.

[24] A. Telikapalli. Virtex-II Pro FPGAs: The platform for programmable systems has arrived. *Xcell journal*, 1(42):10–13, Spring 2002.

[25] Xilinx. *XC4000E and XC4000X Series Field Programmable Gate Arrays. Data sheet*. Xilinx, 1999.

[26] Xilinx. *Virtex 2.5V Field Programmable Gate Arrays. Data sheet*. Xilinx, 2000.

[27] Xilinx. *Virtex-II Pro Platform FPGAs. Data sheet*. Xilinx, 2002.

[28] R. Zimmermann. *Lecture Notes on Computer Arithmetic: Principles, Architectures, and VLSI Design*. Swiss Federal Institute of Technology. Integrated Systems Laboratory, Zurch, Switzerland, 1999.