INTEGRATION, the VLSI journal 42 (2009) 486–503

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Designing coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data-paths ☆

Sotiris Xydis *, George Economakos, Kiamal Pekmestzi

School of Electrical and Computer Engineering, National Technical University of Athens, Iroon Polytexneiou 9, GR 15780 Athens, Greece

ARTICLE INFO

Article history:
Received 30 November 2007
Received in revised form
22 December 2008
Accepted 23 December 2008

Keywords:
Coarse-grain reconfigurable architectures
Flexibility inlining
Canonical interconnection
Carry-save arithmetic
Chain addition
Array multiplier

ABSTRACT

This paper introduces a design technique for coarse-grained reconfigurable architectures targeting digital signal processing (DSP) applications. The design procedure is analyzed in detail and an area-time-power efficient reconfigurable kernel architecture is presented. The proposed technique inlines flexibility into custom carry-save (CS) arithmetic datapaths exploiting a stable and canonical interconnection scheme. The canonical interconnection is revealed by a transformation, called uniformity transformation, imposed on the basic architectures of CS-multipliers and CS-chain-adders/subtractors. Experimental results including quantitative and qualitative comparisons with existing reconfigurable arithmetic cores and exploration results of the proposed reconfigurable architecture are provided.

1. Introduction

The advent of reconfigurable computing [1,2] has generated a whole new research field in the area of digital design. The new computational model augments the available logical density of the circuits, by binding the spatial (high parallelism) and the temporal (high programmability–flexibility) models. Along with the high integration densities provided by the current application-specific integrated circuit (ASIC) technologies, we are able to design dynamic configurable hardware systems on a single chip (configurable system-on-chip, CSoC) [3–6].

A significant number of reconfigurable architectures have already been proposed, varying mostly on the granularity's degree. An overview of the most popular reconfigurable architectures can be found in [2]. Fine-grained architectures [2, Chapter 1; 7,8] such as classical field programmable gate arrays (FPGAs), favor bit-level operations and mapping universality, but suffer from high reconfiguration delays and power consumption. Coarse-grained architectures [2, Chapter 2; 3–5], eliminate the disadvantages of fine-grained ones and preserve universality at most cases. However, they operate only on word-length data formats. Recently, hybrid architectures, which try to combine the benefits of the two above approaches, have been proposed [9,10]. All these solutions propose new architectures which enable dynamic hardware reconfiguration.

In the field of coarse-grained reconfigurable architectures, two major trends have been reported: (1) Coarse-grained reconfigurable arrays [3,12,14] which provide operation level reconfigurability at the cost of low utilization of the underlying hardware. (2) Reconfigurable arithmetic units (RAUs) [16–18] which provide only subword reconfigurability to arithmetic datapaths when data with lower precision are available. This paper extends the two aforementioned major trends by introducing operation level reconfigurability in arithmetic datapaths which incorporate a significant degree of computation density (i.e., array multipliers).

Our work has a twofold aim: (1) to present the techniques which enable operation level reconfigurability to be efficiently incorporated into custom arithmetic datapaths and (2) to apply these techniques in order to introduce an area, time and power-efficient reconfigurable architecture, considering the digital signal processing (DSP) application domain. Specifically, the proposed techniques incorporate flexibility by mapping together the behaviors of a carry-save (CS) multiplier, a CS adder and a CS subtractor onto a stable interconnection scheme. The introduced architecture is a coarse-grain reconfigurable datapath which mainly targets ASIC implementation technologies. It provides fast implementations for the set of mapped operations due to the CS-based logic, which eliminates the time consuming carry propagation. The fast arithmetic operations and the stable interconnection

☆ A preliminary version of this work has been presented in International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS'07), Samos, Greece, July 2007.

* Corresponding author.
E-mail addresses: sxydis@microlab.ece.ntua.gr (S. Xydis), geconom@microlab.ece.ntua.gr (G. Economakos), pekmes@microlab.ece.ntua.gr (K. Pekmestzi).

0167-9260/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2008.12.003

scheme increase the opportunities for operation chaining which has been proved a beneficial technique for DSP algorithms [11,12].

The main contributions of this paper are summarized in the following lines:

(1) The flexibility inlining technique is introduced. It is enabled through a transformation at the register transfer level (RTL) of abstraction, named uniformity transformation. Uniformity transformation reveals an interconnection scheme that remains stable and canonical between the configurations of CS-chained adders, CS-chained subtractors and CS-array multipliers.
(2) An optimized unified cell (UC) that combines the behaviors of multiplication, addition and subtraction, is presented.
(3) Based on the appliance of flexibility inlining, we present a novel coarse-grained reconfigurable architectural template and we analyze it in detail, at the circuit level, based on a specific architecture instantiation.
(4) Evaluation of the proposed architecture is conducted through extensive experimentation and explorative results.

The effectiveness of our approach is proven through a rich set of experimental data (Section 6). Qualitative comparisons show that the proposed architecture is able to handle a large set of mapped arithmetic behaviors without wasting computational resources. Quantitative comparisons with dedicated and previously published reconfigurable architectures are also included. Specifically, the proposed architecture delivers gains up to 46.9% in area coverage, 296.5% in clock frequency, 25.2% in power dissipation and 32.75% in mega operations per second (MOPS) comparing to the reconfigurable multiply-accumulate (MAC) unit in [17]. In comparison to dedicated MACs, the proposed architecture delivers gains up to 218.6% in MOPS and 163.33% in energy efficiency (MOPS/mW). Finally, exploration results, altering the reconfigurability/flexibility incorporated into the proposed architecture, show an almost linear scaling behavior with regard to area complexity and power consumption.

The rest of the paper is organized as follows. Section 2 refers to the relative work about coarse-grained reconfigurable array systems and about non-platform-specific reconfigurable arithmetic cores targeting DSP applications. Section 3 concerns the flexibility inlining technique. Section 3.1 provides the problem description and discusses the motivation of Flexibility Inlining. The uniformity transformation applied on the multiplier's, adder's and subtractor's circuits, exposing the opportunity of stable interconnection scheme between the data-paths of these operators, is described in Section 3.2. In Section 3.3 the design of the arithmetic datapath's basic cell is presented. A novel reconfigurable architecture template is presented in Section 4 and in Section 5 the configurability of the proposed architecture is analyzed. Section 6 presents the experimental results and finally, Section 7 concludes the paper.

2. Related work

Many coarse-grained reconfigurable architectures have been proposed in the literature. The Morphosys reconfigurable system is a complete reconfigurable SoC implemented and optimized at the layout level [13]. It incorporates a 32-bit RISC processor and a 8×8 array of coarse-grained reconfigurable cells for efficient mapping of data-parallel applications [3]. The basic reconfigurable cell consists of an arithmetic logic unit (ALU) and a CS-array multiplication unit. The reconfigurable SoC also incorporates a DMA-controller and a frame buffer for fast data transfers between the memory and the reconfigurable array module. The ADRES architecture [14] proposes a similar reconfigurable array computational platform tightly coupled with a very large instruction word (VLIW) embedded processor.

In [12], a coarse-grain reconfigurable architecture, which targets DSP applications, is proposed. It enables efficient template-based operation chaining. Every node of the applications' data-flow graph (DFG) is mapped on a computational resource, which consists of four ALUs and four multiplication units. The templates are implemented by interconnecting appropriately a number of computational cells. Template-chaining is performed by using a flexible inter-template interconnection network. Although the proposed architecture seems to have performance gains in comparison with the straightforward template-based methods [11], the area overheads imposed by the basic template cell architecture are not negligible.

The aforementioned approaches comprise arrays of computation universal cells enabling the mapping of all the desired behaviors. However, this is achieved in expense of large hardware area overheads and low utilization of the computational resources, since only one computational component (i.e., the ALU or the multiplier) is used in each control step. On the contrary, a lot of research has been moving towards domain-specific reconfigurable arithmetic cores. Domain-specific reconfigurability includes the idea of providing only a certain amount of reconfiguration capability to the developing ASIC, according to the desired applications' domain specific needs. Characteristic examples of domain specific reconfigurable architectures are the following.

Morphable multipliers proposed in [15] are multi-mode (morphable) functional units (MFU) based on the exploitation of the compressors' timing slack in tree multipliers. The system is configured to perform two-operand multiplication or addition. Each desired mode is analyzed separately and the adder chains, which will form the adder mode, are produced. However, only sharing between single addition and multiplication operations is considered, so the applicability set of the produced designs is limited.

In [16] a new decomposition technique of the partial product matrix was presented along with the design of a reconfigurable parallel inner product processor at the transistor level, using both traditional binary and pass-transistor logic [22] for its components. In [17] a reconfigurable MAC architecture based on the decomposition scheme of [16] was proposed. Using reconfiguration, the architecture trades bitwidth for array size and due to an extra allocated arithmetic selection unit is able to operate on various data formats.

In [18] a multiplier of variable precision which adapts itself at run-time to different data word-lengths is introduced. The multiplicative scheme is based on the weighted addition of shifted partial products which suits well on classical FPGA devices. However, this approach is optimized only for FPGA-target specific platforms. In [19] the authors presented a reconfigurable multiplier implemented on FPGA targeting mostly to the cryptographic applications. They actually partition a radix-4 folded multiplier array horizontally, keep only the first partitioned segment and feedback the intermediate results in order to complete the multiplication operation.

The aforementioned schemes for reconfigurable multipliers/MAC units (except the scheme in [15]) target one type of operation and exploit the notion of reconfiguration only for on-line bitwidth adaptability of the multiplication. The main differentiator in our work is that the proposed architecture targets many types of operations through operation level reconfigurability, so that the same hardware performs a large set of different operations (i.e., addition, multiplication, substraction, chained addition etc.). Compared with the array based reconfigurable architectures,

we differ in the utilization degree of the underlining computational hardware resources, since all the operations are mapped onto a single RAU. In general, this work provides the techniques and the corresponding architectures to enable operation level reconfigurability along with efficient operation level parallelism into custom arithmetic datapaths with large computational resources (i.e., multiplier, chain-adders etc.).

3. The flexibility inlining technique

3.1. Motivation and problem description

The functional generality of most of the reconfigurable architectures introduced, is performed by both interconnection's and functional block's reconfigurability. Classical FPGA structures use switch matrices for programmable routing, while coarse-grained reconfigurable arrays use mostly complex bus-based or highly multiplexer-based interconnections enabling the transfers between the configurable blocks. These complex interconnection schemes impose a significant area overhead and also a reconfiguration delay overhead due to the augmented number of configuration bits needed to control the interconnections.

On the other hand, from an algorithmic point of view, DSP applications are based mainly on four operative modes, namely, addition, subtraction, multiplication and shifting. This can be easily confirmed by profiling the main DSP kernels' DFGs. Thus, the ability of performing the previously mentioned primitive operations, by maintaining temporal flexibility and stable/simple interconnection structure, can expose high area and reconfiguration delay gains comparing with classical coarse-grained architectures. The remainder of this section describes the flexibility inlining technique. The proposed technique enables the mapping of the basic computational intensive DSP operations (addition, subtraction, multiplication) onto the structure of an array multiplier, while maintaining the efficient routing between the initial multiplier's basic cells.

Consider a $N \times N$ array multiplier based on CS-adders [22]. The array consists of N^2 multiplier cells (MCs) and their interconnections. The structure of such a multiplier is given in Fig. 1a for $N = 4$. Let i be the array's line index, and j the array's column index. The overall architecture can be described by the following relations:

$$MC_{i,j} : a_j \times b_i \times si_{i,j} \times ci_{i,j} \rightarrow so_{i,j}, co_{i,j} \tag{1a}$$

$$si_{i,j} = 0, \quad i = 0 \tag{1b}$$

$$si_{i,j} = so_{i-1,j+1}, \quad i \in \{1, N-1\} \tag{1c}$$

$$ci_{i,j} = 0, \quad i = 0 \tag{1d}$$

$$ci_{i,j} = co_{i-1,j}, \quad i \in \{1, N-1\} \tag{1e}$$

$$a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j} \in \{0, 1\} \tag{1f}$$

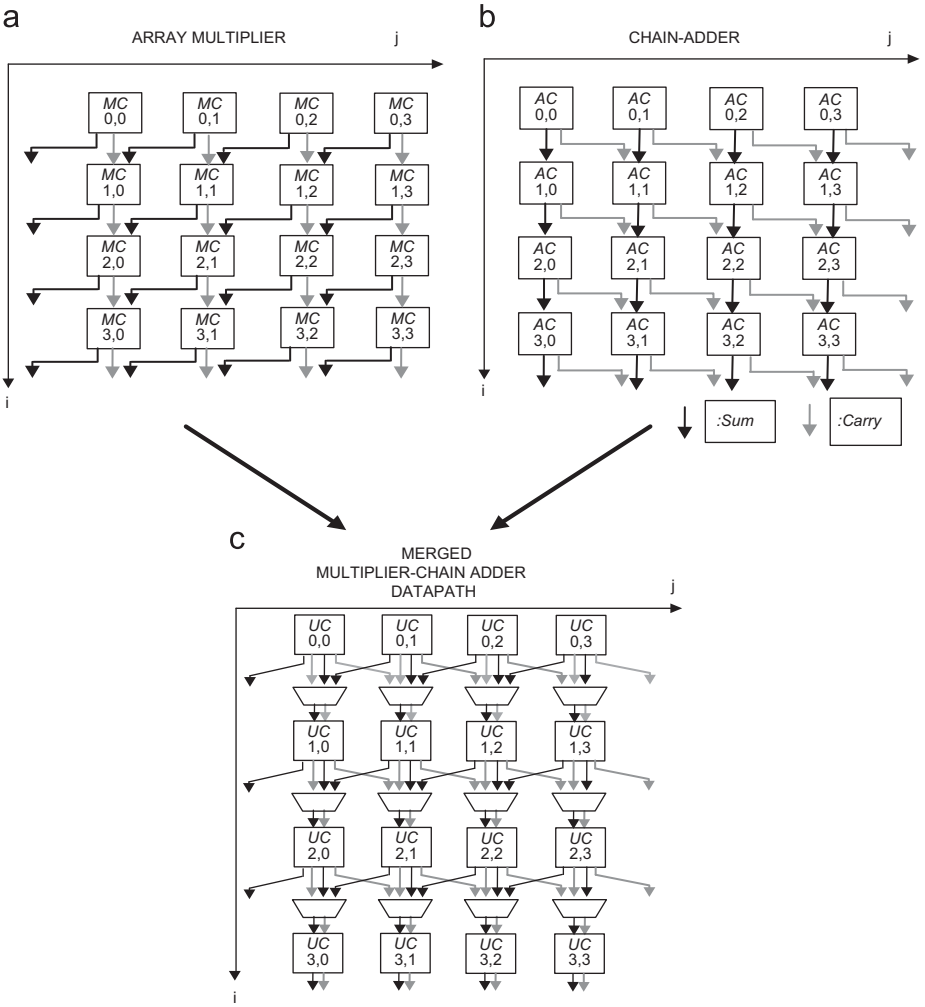

Fig. 1. (a) 4×4 multiplier, (b) 4×4 chain-adder, and (c) straightforward unified architecture.

Relation 1a defines that each basic cell is a multi-input–multi-output function targeting the Boolean domain, as imposed by relation 1f. The order of the binary variables in the specification function 1a also defines the instantiated input/output port order at basic cell level. Relations 1a to 1f refer to the routing properties of the CS-multiplier's architecture. Actually, they determine the source of cell's inputs (s_i, c_i).

The array's structure of CS-adders for chain addition, has significant similarities with the multiplier's one. In both cases N^2 cells are required and the basic component of full addition cell (AC) is present. An illustrative example of 4×4 CS chain-adder is given in Fig. 1b. The corresponding relations that describe the chain-adder structure are the following ones:

$$AC_{i,j} : a_j \times si_{i,j} \times ci_{i,j} \rightarrow so_{i,j}, co_{i,j} \tag{2a}$$

$$si_{i,j} = x_{0,j}, \quad i = 0 \tag{2b}$$

$$si_{i,j} = so_{i-1,j}, \quad i \in \{1, N-1\} \tag{2c}$$

$$ci_{i,j} = y_{0,j}, \quad i = 0 \tag{2d}$$

$$ci_{i,j} = co_{i-1,j-1}, \quad i \in \{1, N-1\} \tag{2e}$$

$$a_j, x_{0,j}, y_{0,j}, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j} \in \{0, 1\} \tag{2f}$$

Relation 2a defines each basic cell's Boolean function, Eqs. (2b)–(2e) refer to the routing properties of the structure and the ordered binary variables in the specification function 2a define the instantiated input/output port order at basic cell level. The same relations also apply in chain-subtraction, with the only difference in the basic cell's truth table.

Comparing Fig. 1a with Fig. 1b, the routing differences between the two structures are exposed. The routing scheme, for both architectures, is denoted by relations (1c) and (1e) for the multiplier and by relations (2c) and (2e) for the chain-adder, respectively. Actually, these relations are depicting the dependence of a cell with the cells located on the previous row and the neighbor columns. Relation (1c) states that the multiplier's input *si* of $MC_{i,j}$ results from the cell's output *so* of the previous row $(i-1)$ and the right column $(j+1)$. Respectively, relation (1e) denotes that input *ci* results from the cell's output *co* of the previous row $(i-1)$ and the same column (j). In the chain-adder's case, relations (2c) and (2e) state that $AC_{i,j}$ receives its input signal *si* from the cell's output *so* of the previous row $(i-1)$ and the same column (j), while the input signal *ci* is the cell's output *co* of the previous row $(i-1)$ and the left column $(j-1)$.

Finally, let us consider a UC (Fig. 1c) that is able to function as MC or as AC or as subtraction cell (SC). The function of the cell depends on a set of appropriate selection signals which control the operating mode of the UC. An optimized implementation of the UC is presented at Section 3.3. The number of inputs and outputs for the UC derived by the union of the basic cells' input ports. In our case, the port declaration of the UC is going to be the following:

$$\begin{aligned} &Ports_{MC} \cup Ports_{AC} \cup Ports_{SC} \\ &\quad = \{a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &\quad \cup \{a_j, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &\quad \cup \{a_j, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &\quad = \{a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \end{aligned} \tag{3}$$

As stated previously, mapping together multiplication, addition, subtraction, chain addition and chain subtraction operations on a single circuit, and changing dynamically the configuration between these behaviors can lead to a high performance functional unit, especially for the DSP domain. However, the combinative mapping of the above configurations in a straightforward manner, Fig. 1c, suffers from the described routing dissimilarity, and from the functional dissimilarity occurring between each basic cell configuration context. Routing dissimilarity imposes a complex interconnection scheme between the cells. Actually, a 2 to 3 switch circuit is needed for every $UC_{i,j}$ to route appropriately the cell's outputs. This scheme is area inefficient, augments the architecture's configuration word-length and imposes a highly complex interconnection routing between the cells (Fig. 1c).

The flexibility inlining technique tackles the aforementioned routing and functional dissimilarities through (1) a transformative procedure which is named uniformity transformation, and (2) a design procedure for optimized implementation of the UC. Uniformity transformation (Section 3.2) is applied on the overall architecture (primary inputs' bit-order, UC's input and output ports) and results to a stable routing scheme among the desired arithmetic behaviors. The UC's design procedure (Section 3.3) concerns the internal logic of the UCs, in order to maximize hardware sharing among the different configurations.

3.2. Uniformity transformation: Enabling common interconnections among arithmetic configurations

Uniformity transformation is completed in three stages, analyzed in Sections 3.2.1, 3.2.2, 3.2.3 and shown in Fig. 2, respectively. Uniformity transformation should not be confused with the uniformization methods presented in the field of systolic arrays [20,21]. Uniformization is imposed on linear recurrence equations (inter-loop dependencies), which describe the system's behavior. It transforms linear into uniform recurrences in order to enable a direct mapping on a locally connected systolic array. The proposed uniformity transformation is imposed on the bit-level description of arithmetic datapaths in order to provide an efficient datapath to datapath mapping. In an abstract manner, it can be stated that the main goal of both approaches is the generation of constant structures (inter-loop dependencies in uniformization, interconnection schemes in uniformity transformation) between descriptions with non "one-to-one" correspondence.

3.2.1. Selection of the reference architecture

At the first stage, a reference architecture is selected. The candidate architectures are actually two, since the chain-adder and the chain-subtractor have the same interconnect structure. The routing scheme of the reference architecture will remain stable till the end of the process. The two routing architectures are equally canonical and enable efficient VLSI implementation. So, the final selection is driven by the utilization degree of each operation mode. For example, if one application requires a large number of multiplication operations, it is preferable to select the multiplication scheme as reference architecture. Equally, if the additions or subtractions occur more often, then it is more serviceable to select the chain-addition architecture. The utilization degree for each operation can be extracted by profiling the dataflow-graphs of the applications which will be mapped on the proposed reconfigurable arithmetic datapath. Without loss of generality, the reference architecture for the undergoing analysis is considered the chain-addition's one.

3.2.2. Generation of the mirror-symmetrical architecture

Given that the reference architecture has the routing scheme of a CS-chained-adder, in the subsequent stage of the transformation

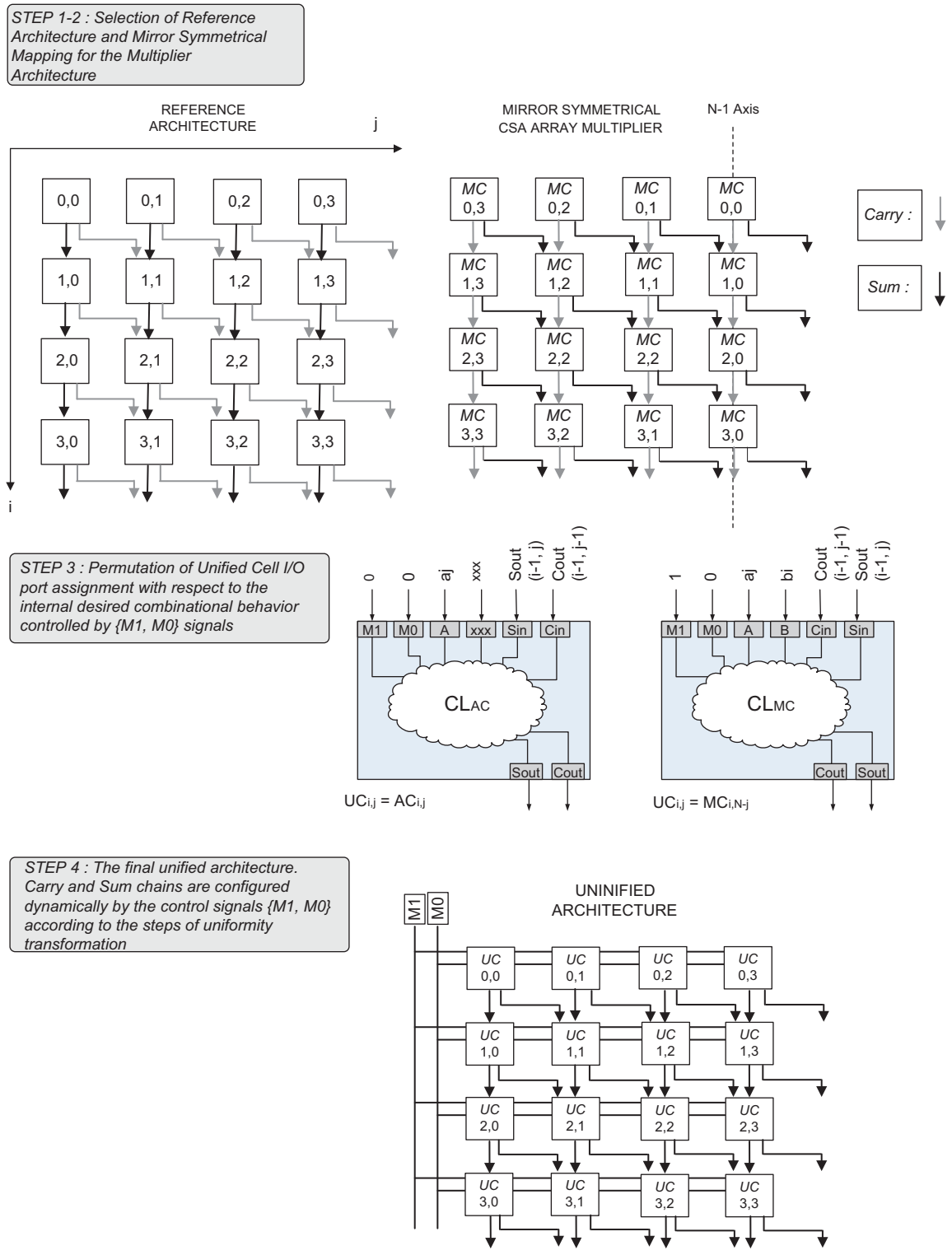

Fig. 2. The uniformity transformation.

we reverse the bit-order of the multiplier's input $a_j, j \in [0, N-1]$, and the mapping order of the basic cells to preserve the correct functionality (*STEP 2* in Fig. 2). The resultant architecture is mirror-symmetrical to the initial one, over the $j = N-1$ vertical axis. The new relations describing the mirrored architecture are obtained by substituting $j = (N-1) - j$ and by rewriting the routing equations for the multiplier cell. The reference architecture remains the same. The new form of $MC_{i,N-j}$ is given below

$$a_{(N-1)-j} \times b_i \times si_{i,(N-1)-j} \times ci_{i,(N-1)-j} \rightarrow so_{i,(N-1)-j}, co_{i,(N-1)-j} \quad (4a)$$

$$si_{i,(N-1)-j} = 0, \quad i = 0 \quad (4b)$$

$$si_{i,(N-1)-j} = so_{i-1,(N-1)-j+1}, \quad i \in \{1, N-1\} \quad (4c)$$

$$ci_{i,(N-1)-j} = 0, \quad i = 0 \quad (4d)$$

$$ci_{i,(N-1)-j} = co_{i-1,(N-1)-j}, \quad i \in \{1, N-1\} \quad (4e)$$

$$a_{(N-1)-j}, b_i, si_{i,(N-1)-j}, ci_{i,(N-1)-j}, so_{i,(N-1)-j}, co_{i,(N-1)-j} \in \{0, 1\} \quad (4f)$$

The routing relations of the new multiplier scheme have changed. By posing $k = N - j$, relation (4c) becomes $si_{i,k-1} = so_{i-1,k}$, while relation (4e) becomes $ci_{i,k-1} = co_{i-1,k-1}$. These new relations indicate that the input signals for both chain-adder and mirrored-multiplier cell structures are issued by the same cells of the previous row. Therefore, the routing dissimilarity constraint is partially relaxed, since the input signals now have the same sources. However, each source is still attached to different ports on the sink cell, considering the multiplier's and the adder's basic cell. The different port attachment imposes the need for two multiplexors 2 to1 under each cell in order to drive the inputs in a proper way. The elimination of the intermediate multiplexors is performed by the third stage of the uniformity transformation.

3.2.3. Permutation on UC's internal port map

Let us consider a unified basic cell structure. A straightforward port assignment generates the need of the internal multiplex usage. This happens because port *si* in the chain-adder's case has input from the above cell, but in mirrored-multiplier's case it has inputs from the left above cell. An equivalent condition holds for port *ci*.

To overcome this limitation, we propose to keep steady the UCs external port interface and alter the intra-cell port mapping. The altering can be made as a simple permutation operation on the UC's internal port map. So, let us assume that the port interface template of the UC is the binary vector $\{I_1, I_2, I_3, I_4, I_5, I_6, O_1, O_2\}$, where I_i stands for the inputs and O_j for the outputs respectively.

In case of adder/subtractor mode the internal port map is $\{M_1, M_0, a_j, xxx, \underline{si_{i,j}}, \underline{ci_{i,j}}, co_{i,j}, so_{i,j}\}$. The *xxx* input port is not used because three input ports are sufficient for the adder case. In case of multiplication mode the internal port map follows the template $\{M_1, M_0, a_j, b_i, \underline{ci_{i,j}}, \underline{si_{i,j}}, co_{i,j}, so_{i,j}\}$ with respect to, in both cases, the external port declaration of the reference architecture. The b_i input port is bound to the vertical's multiplicand bit.

The permutation over the internal port map eliminates the need of intermediate multiplexors by properly driving the input and output signals of each cell. It is performed at design time and actually pre-routes the input and output ports of each UC to its internal combinational logic (*CL*). The *CL* operates on the pre-routed wires and computes the proper values according to the control signals $\{M_1, M_0\}$. The internal semantic values of the two rightmost input and the two output ports of the UC alter when the UC is configured as multiplication or addition/subtraction cell (*STEP 3* in Fig. 2). However, the external routing scheme for the whole architecture is preserved stable.

The three stages form the proposed uniformity transformation. A stable and canonical routing scheme is exposed for efficient mapping of multiple arithmetic behaviors, on the same architecture, minimizing the routing complexity and circuits' criticality imposed by interconnection dissimilarities (Fig. 2, *STEP 4*). The feasibility of multiple behavior mapping generates opportunities of flexible, run time configurable and efficient arithmetic units.

3.3. The UC structure

The UC is actually a mapping of the desired behaviors into a single module. The problem of mapping together multiple behaviors onto the same circuit has been stated in [23,24]. Given the dataflow graphs of the behaviors, they try to minimize area complexity by efficiently reusing the allocated resources. These techniques operate at a more abstract level and are based on hardware sharing of basic RTL components. In order to design efficiently the UC, we have to work on a lower level of abstraction (gate level).

In a straightforward implementation of the UC, Fig. 3a, the arithmetic behaviors are instantiated as sub-modules (AC: addition basic cell, SC: subtraction basic cell, MC: multiplication basic cell) into the same top module to form the UC. The desired behavior of the UC results from proper control signals $\{M_1, M_0\}$ which drive an output multiplexer. However, this straightforward design methodology introduces a large area overhead with a low utilization degree of the overall circuit of each cell. Large area overheads are introduced because of the un-exploited common gate-level datapaths, which exist inside the module. The main reason that this happens is that this implementation does not take into account the inter-cell sharing optimizations at the logic level.

To overcome this inefficiency, we followed a design procedure which optimizes the hardware sharing among the behaviors mapped on a UC. The proposed methodology is based on the ESPRESSO logic minimization tool [25] and on the output phase optimization technique presented by Sasao [26].

The output phase optimization technique searches the design space of a multi-output Boolean function in order to find the set of the outputs' phases which minimize the needed product terms. The minimization of the product terms of a Boolean function leads to smaller area circuitry.

The ESPRESSO tool incorporates the utility of performing output phase optimization as a user specified parameter. We performed exhaustive search granted by the fact that the UC has a small design space. In all cases, we considered the permutation over the UC internal port map, as imposed by the last step of the uniformity transformation.

The design of a UC consists of four stages. At first, the truth tables of the desired cell's behaviors/configurations are formed in PLA format. In our case, three truth tables are formed. The first one describes the Boolean function of the multiplier cell, the second one describes the adder's cell function and the third one the subtractor's cell function.

The next step is the concatenation of the truth tables. A proper number of control bits is introduced to form a single unified truth table for all the expected cell's configurations (MC, AC, SC). The unified truth table aggregates all the configurations in a single multi-output function, so that the common gate datapaths between the desired behaviors are revealed to the ESPRESSO tool, maximizing the gate sharing among configurations.

The unified truth table is supplied at the ESPRESSO tool which outputs the overall minimized logic expression. As it was mentioned, the ESPRESSO tool is properly tuned to perform logic minimization with respect to the output phase optimization technique. The minimized output is still in PLA format. An equivalent circuit can be formed in RTL so as the functional correctness is tested.

The synthesis of the cell can be performed using either the minimized PLA or its RTL equivalent. In order to prove the efficiency of our approach in designing the UC internal structure, we synthesized the UC's configurations for the cases of (a) a straightforward implementation (Fig. 3a) with no sharing, (b) a straightforward implementation (Fig. 3b) in which the AC and the SC are combined together (AC/SC sharing), (c) a sub-optimized UC using custom RTL components (full sharing at RTL level, Fig. 3c) and (d) a UC based on the described technique (sharing at gate level, Fig. 3d), using the Synopsys synthesis toolsuite [32]. The synthesis was made according to the standard cell design methodology using the TSMC 0.13 um technology library [34]. All the configurations were synthesized under the timing constraint of 0.4 ns for the UC's propagation delay. The comparative results are reported in Table 1. Our methodology delivers area savings of 71.6% and power savings of 79%, comparing with the first straightforward approach (UC_{STRFW1}). In comparison with the second straightforward approach (UC_{STRFW2}), in which the AC and

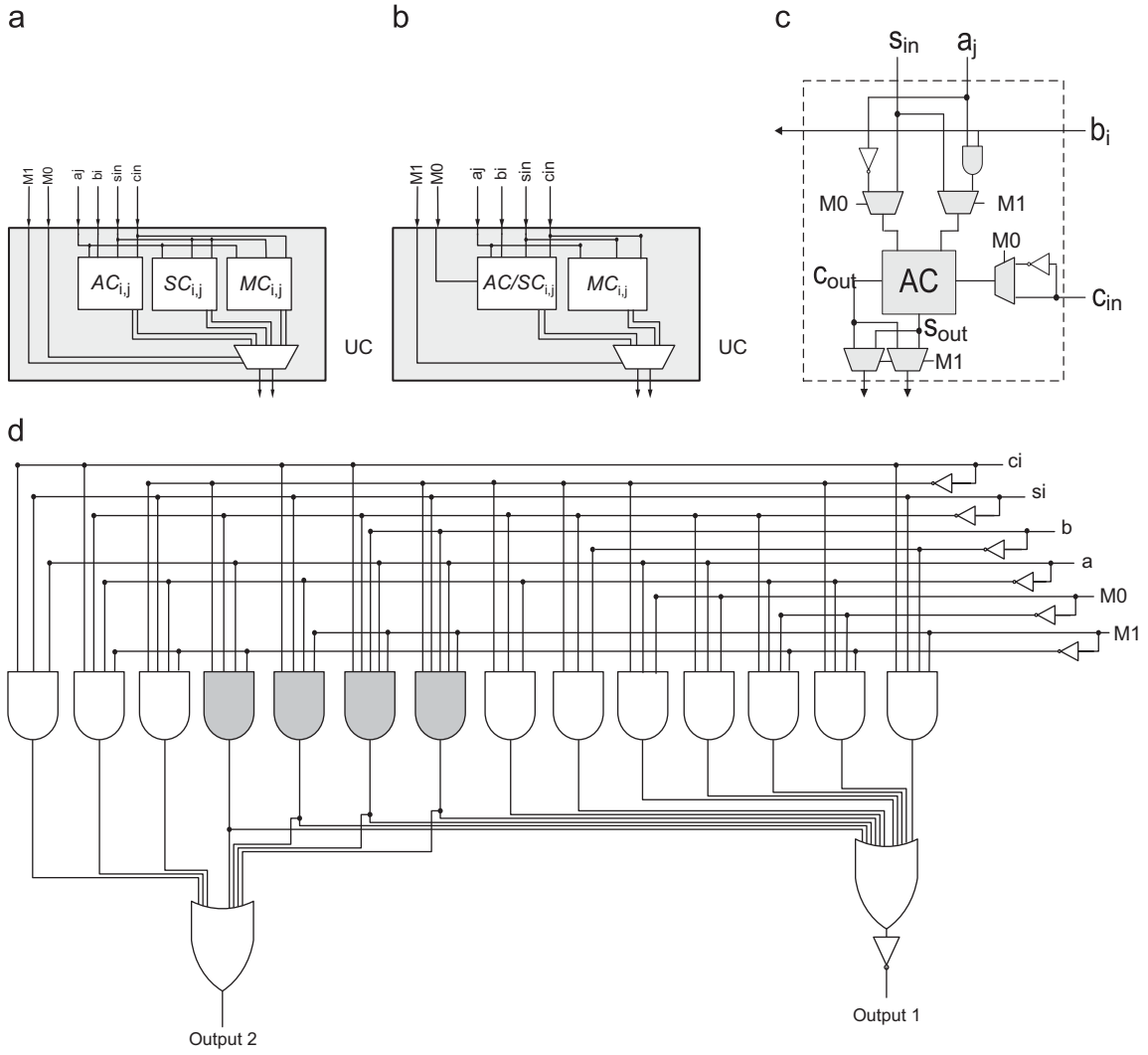

Fig. 3. Several implementations of the UC: (a) straightforward with no sharing, (b) straightforward with AC/SC sharing, (c) optimized at the RTL, and (d) optimized with the proposed technique.

Table 1
Comparison of the UC's implementation strategies.

Cell	Area (um^2)	Power (uW)	Area overheads (%)	Power overheads (%)
UC_{STRFW1}	370.2	271.8	71.6	79
UC_{STRFW2}	275.1	255.3	27.5	68.2
UC_{SUBOPT}	232.6	213.9	7.8	40.9
UC_{OPT}	215.7	151.8	0	0
MC	93.4	63.2	−56.7	−58.3

SC are shared, area and power gains of 27.5% and 68.2% are reported, respectively. Compared with the sub-optimized RTL implementation of the UC, the savings in terms of area are up to 7.8% and in terms of power up to 40.9%. In Table 1, we have also included the synthesis results of the simple multiplier's cell (MC) to indicate the overheads imposed of the optimized UC in comparison with the standard multiplier cell. The UC requires approximately double area and dissipates double power, compared with the MC cell. However, the UC can be configured either as multiplication cell, or as AC or as SC.

4. Reconfigurable kernel architecture

Based on the flexibility inlining technique, a reconfigurable kernel architecture template was extracted. An abstract model of the reconfigurable architecture template is illustrated in Fig. 4.

The reconfigurable architecture has been designed to operate on operands of 16-bit word width, since such a word-length is considered adequate for the majority of the DSP datapaths [12]. It is composed by (1) the reconfigurable pipeline stages of UCs, (2) the pipeline registers between each stage, (3) the register bank, (4) the external interconnection network, (5) an arithmetic conversion module and (6) the configuration register which drives the overall architecture. The internal structure of each reconfigurable pipeline stage is formed by an array of UCs which adopts the canonical interconnection scheme revealed by the uniformity transformation (Section 3.2). Multiplications are mapped in a pipelined way through the reconfigurable pipeline stages. Single or chained additions/subtractions can be performed in one control step inside each reconfigurable pipeline stage. The number of reconfigurable pipeline stages could be larger or smaller than four, according to the operation level parallelism that the designer seeks to achieve. The rest of the paper assumes four pipeline stages, which form a balanced solution between the

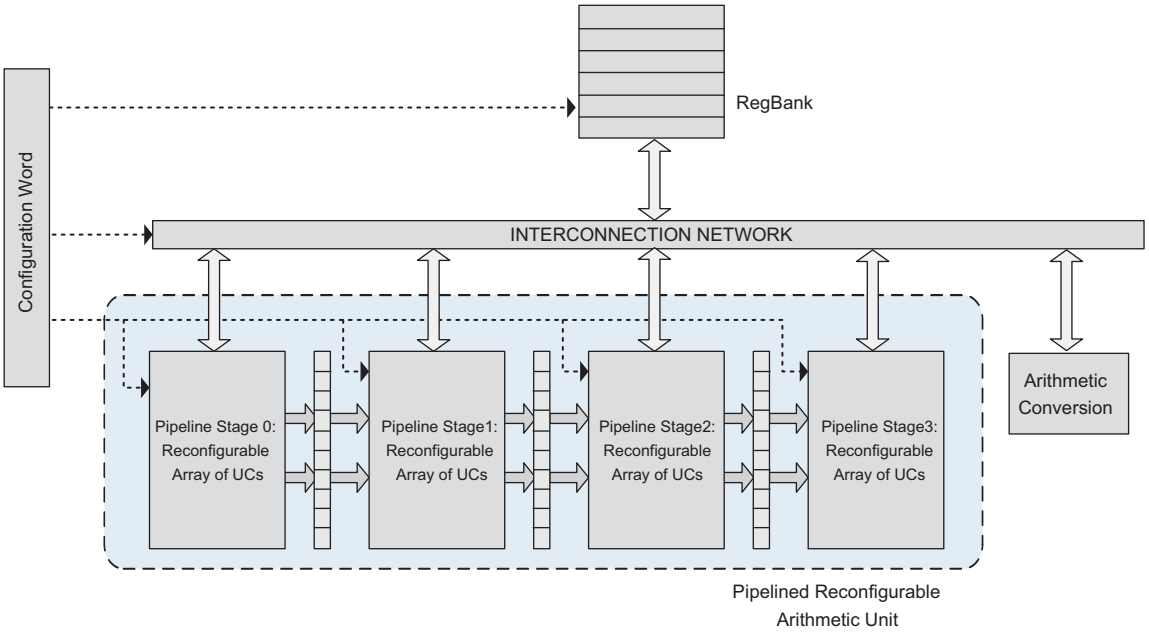

Fig. 4. Abstract model of the reconfigurable kernel architecture.

available operation level parallelism and the number of pipeline stages that are provided by the overall architecture. The register bank is used for the storage of intermediate results. The external interconnection network performs the communication either between the register bank and the reconfigurable pipeline stages or between non-adjacent reconfigurable pipeline stages. The arithmetic conversion module performs conversion from the CS format to the conventional binary whenever it is needed.

In the rest of this section, further analysis of the detailed description of the proposed architecture's datapath is given. This analysis is based on a specific architecture template instantiation. Alternative implementations/instantiations of the architecture are explored in Section 6.3.

An instantiation of the abstract model architecture is depicted in Fig. 5. The kernel comprises (1) a pipelined 16×12 array of UCs along with a 16×4 array of multiplier's cells (the pipelined RAU), (2) a multiplexer-based interconnection network, (3) a small number of registers for intermediate variables' storage (the RegBank), (4) a CS to conventional binary arithmetic conversion module, (5) a context register and 6) three reversing-order modules (circuit level modules not shown in the abstraction of Fig. 4). The UCs inside each reconfigurable pipeline stage are interconnected according to the canonical scheme of the reference architecture. In this paper, without loss of generality, we consider the CS chain adder scheme as reference architecture (Section 3.2.1).

The RAU array consists of four pipeline stages. Each pipeline stage can operate either as a completely independent flexible arithmetic datapath or in coordination with the above stage. When independent execution is performed, the instantiated kernel emulates a multi-output functional unit. The coordinated execution takes place when the pipeline's datapath operates on the previous stage's outputs. The circuit inside each stage is fully combinational enabling efficient operation chaining of the mapped behavior. The RAU can be clocked by high frequency clocks (Section 6) because of its pipelined structure and its CS internal architecture.

Fig. 6 depicts the internal structure of the first two pipeline stages of the RAU. The reconfigurable pipeline stages have been designed with a degree of heterogeneity. Pipeline stage 0 (Fig. 6) includes an intermediate line of 2 to 1 multiplexors (above the third row), which enable this stage to perform in parallel two independent CS additions or subtractions of two, three ($O0$ in Fig. 6) or four operands ($O1$, $O3$ in Fig. 6). In contrast, pipeline stage 1 contains the multiplexer intermediate line at the beginning of each stage so as to route properly the pipeline's inputs. Also, it includes only one output port (i.e., $O7$ in Fig. 6). The third reconfigurable pipeline stage, which is not included in Fig. 6, has the same internal structure as pipeline stage 1. The three pipeline stages consist of UCs and are run-time reconfigurable. The last pipeline stage needs no configuration as it consists only of gated adders (Fig. 7a), the basic component of array multiplier cells (MCs). It is used only as the final stage of a 16×16 multiplication. This design decision reduces the overall hardware area and the required wiring of the RAU.

The reversing-module (Fig. 7b) consists of a simple hardwired 16×16 reversing-order crossbar which assigns correctly the input data to the UCs when the overall kernel is configured to perform a multiplication. The two wire crossbars, which are allocated in the bottom of the last pipeline stage, target the highest order carry and sum bits respectively, after the completion of a 16×16 multiplication operation. Thus, they enable the final addition to be performed with no extra cycle loss.

The arithmetic conversion module (CS2Bin) implements the conversion of CS to conventional binary format. The circuitry for arithmetic conversion consists of a carry skip adder/subtractor [22]. The two-operand carry-skip adder/subtractor contributes to the result's conversion from the redundant CS arithmetic format to the conventional binary format. The carry-skip adder reduces the requisite time for carry propagation by skipping over groups of 4 consecutive adder bits, without wasting many hardware resources like the carry-look-ahead adders [22]. For further optimization of the carry-skip circuit, the optimized carry-skip adder scheme with low carry-absorb time proposed in [28], can be adopted. The case of overflown results can be handled by the well known techniques referenced in [22,27].

The CS to binary conversion of the multiple CS formatted RAU's outputs forms a potential bottleneck, since only one arithmetic conversion module is allocated. In order to tackle this potential bottleneck without spending additional hardware resources (i.e., an extra allocated CS2Bin module), the instantiated architecture

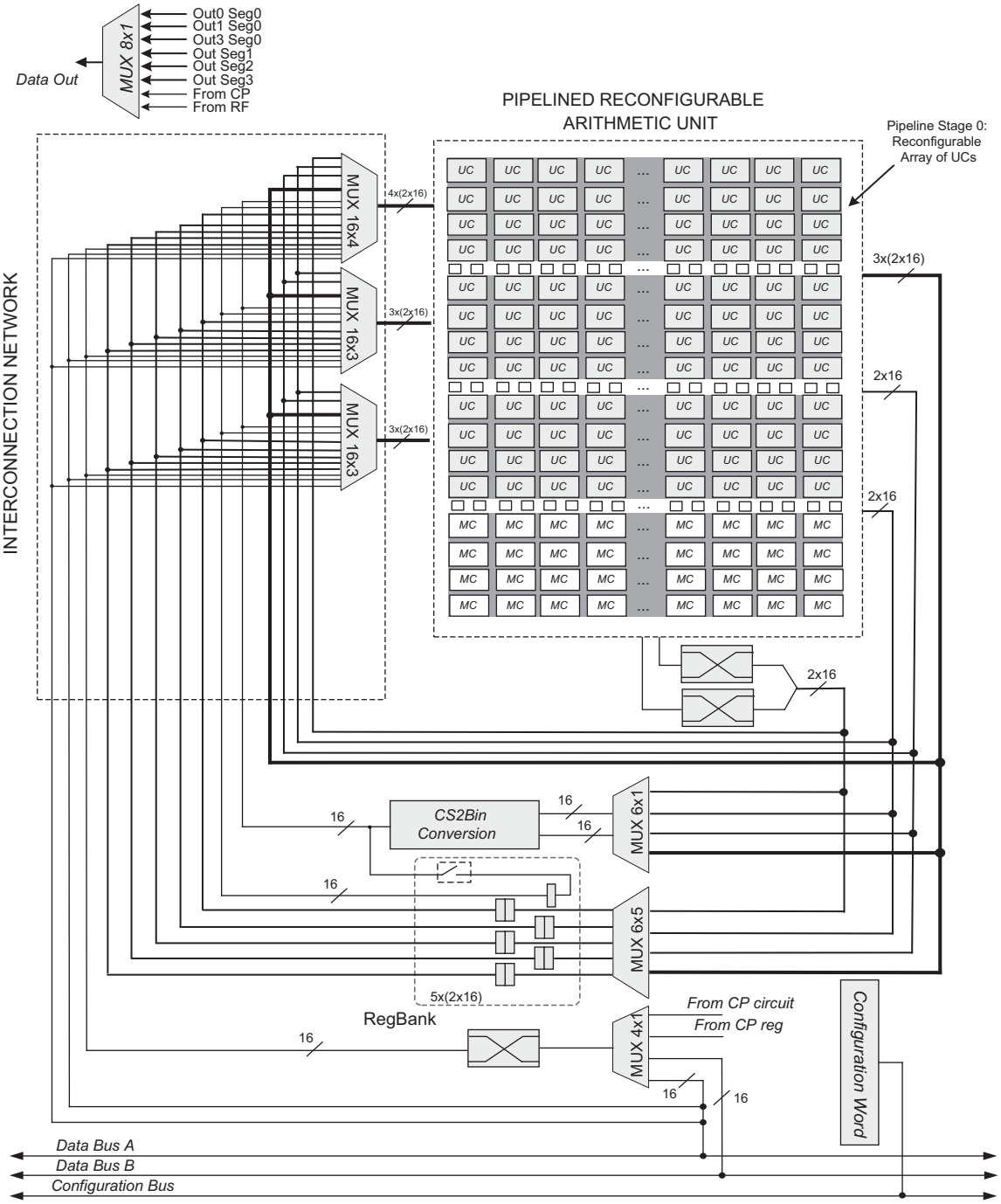

Fig. 5. A detailed instance of the reconfigurable kernel architecture.

template enables arithmetic operations over redundant CS formatted operands. RAU's output bits (in CS format) can be used directly to the next RAU computation without passing from the arithmetic conversion module. The efficient manipulation of this potential bottleneck is also supported by the intra-pipeline stage operation chaining which contributes to less intermediate results. With proper operation scheduling and by continuously supplying the carry-skip adder with CS inputs, the bottleneck can be further relaxed. The combination of the last two strategies contributes to early producing the binary formatted values needed in future control steps.

Taking into account the pipelined organization of the RAU, multiplication operations larger than 16×4 are mapped as a pipelined CS operation. A 16×16 multiplication is completed in 4 clock cycles, with the full precision of a 32-bit final product. The early generated product bits (product's LSBs) are propagated through the stages of the RAU in a pipelined way (Fig. 6). In case that 16-bit product's precision is considered enough, the extra Flip-Flops can be ignored. In Section 6 the measured architectures are able to provide 32-bit precision for the multiplication's product.

The current structure of the reconfigurable kernel architecture requires that every multiplication operation has to be initiated at the first pipeline stage (pipeline stage 0 of Fig. 6) and move along the pipeline stages in the conventional order. The vertical multiplicand (input port b_i of the UC) has to be segmented in

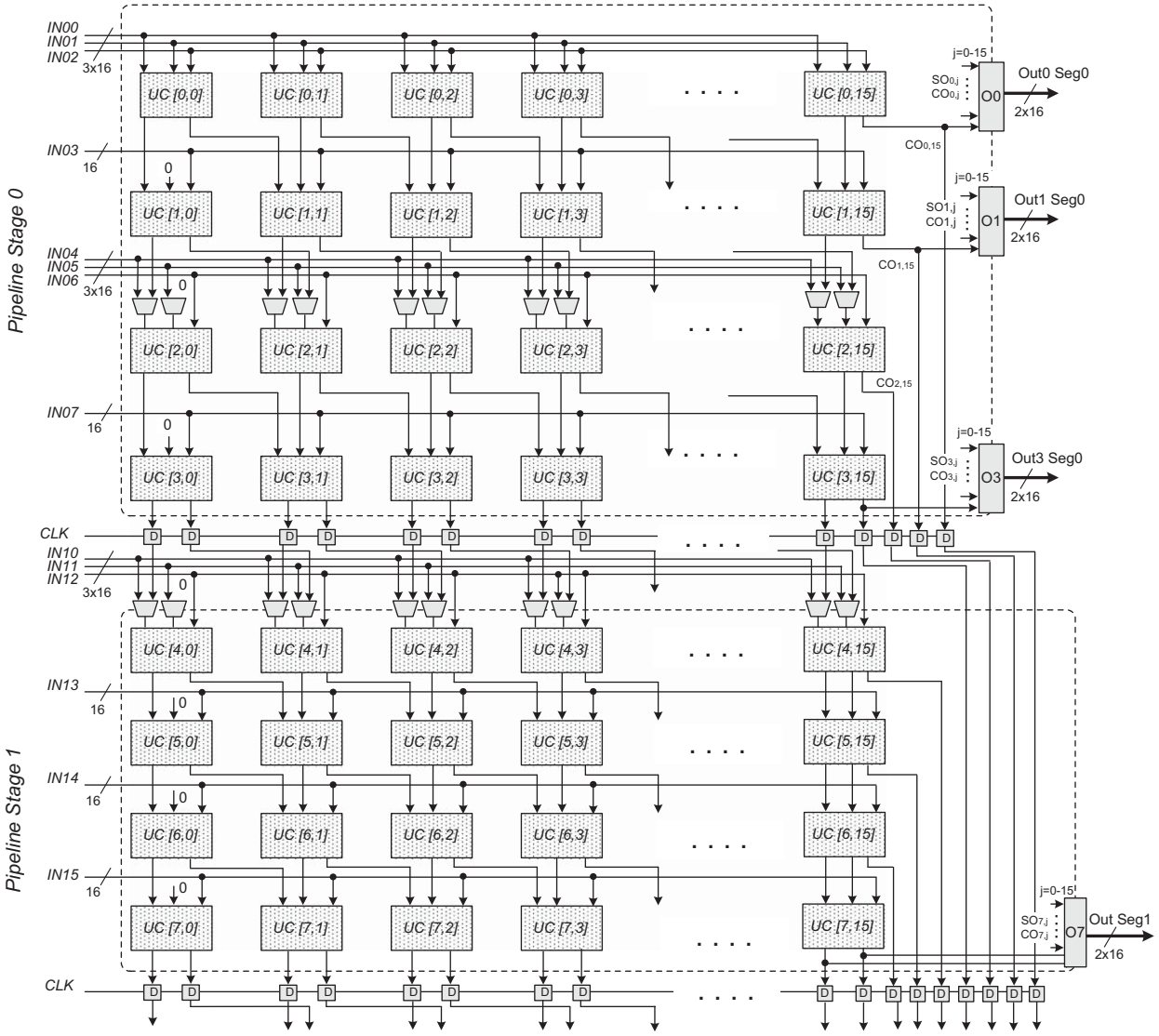

Fig. 6. Internal structure of pipeline stages 0–1. Stage 2 is identical to stage 1. Stage 3 is the typical CS array multiplier scheme.

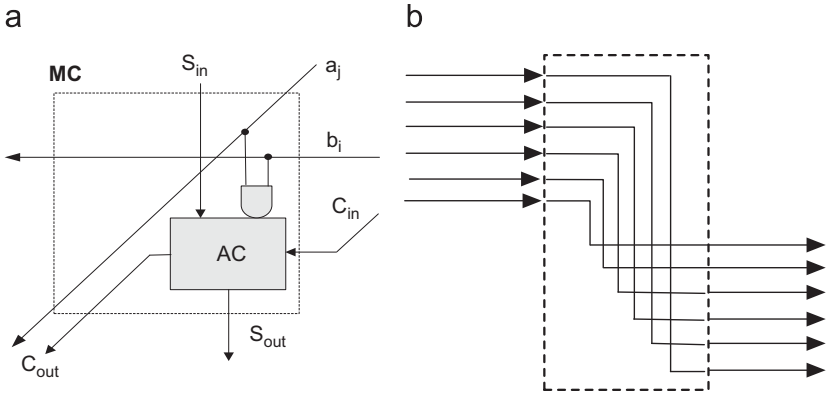

Fig. 7. (a) The gated-adder MC circuit and (b) the reversing module.

four groups of 4-bits and every group has to be issued to its pipeline stage in a synchronized manner. This synchronization mechanism of the vertical multiplicands has not been included in Fig. 5, for clarity reasons. It is illustrated separately in Fig. 8. It is composed of 24 D-Flip-Flops interconnected in a way which permits (1) the segmentation of the vertical multiplicand's word in a cycle by cycle basis and (2) the correct issuing of the segmented words to the proper pipeline stage of the RAU.

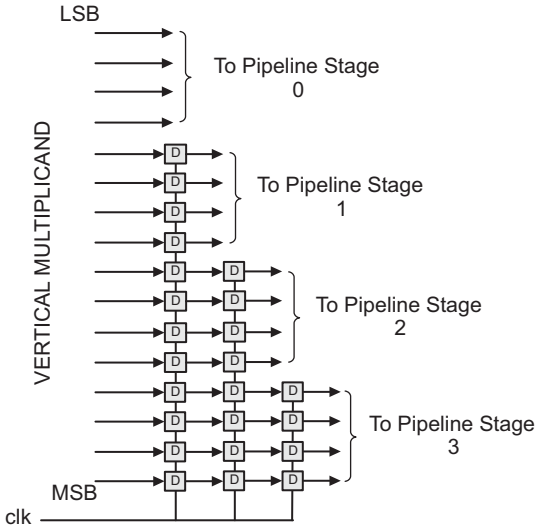

Fig. 8. The synchronization circuit for the vertical multiplicand.

5. Configurability of the architecture

This section describes the configurability issues of the proposed architecture. In order to provide a detailed description, we use the same instantiated architecture template as in Section 4. The following analysis can be applied for every architecture template in a straightforward manner.

The configurability of the instantiated kernel architecture of Section 4 is presented both at the architecture and at the operation level. At the architecture level the RAU is able to operate (1) as one multi-cycle and pipelined functional unit (i.e., as a 4-way pipelined multiplier) or (2) as 4 stand-alone and independent single cycle computational units (adders/subtractors). The RAU's pipelined structure permits high throughput applications to be mapped, while independent execution of each pipeline stage enables the exploitation of the application's inherent operation level parallelism.

Operation level reconfigurability is referred to each pipeline stage separately. Table 2 reports the basic operation modes of each reconfigurable or non-reconfigurable (NR) pipeline stage of the architecture in Fig. 5. Table 2 shows that each pipeline stage is able to operate as an independent processing element which can be configured either (1) as a 16×4 multiplier or (2) as a chain adder–subtractor of up to six binary formatted operands or (3) as a chain adder–subtractor of up to 3 CS formatted operands or (4) as a chain adder–subtractor of a number of hybrid formatted operands (i.e., 2 CS operands and 2 conventional binary operands). Additionally, Table 2 depicts that configurability is not the same for all the four pipelined stages. Thus, the pipelined stage 0 of Fig. 6, which is enhanced by the inter-row multiplexors, exhibits greater configurability compared with the remaining ones. The non-enhanced stages (i.e., pipeline stage 1 in Fig. 6) can be configured either (1) as a chain-adder/subtractor of 6 binary formatted inputs, or (2) as a chain-adder/subtractor of 3 CS formatted inputs, or (3) as a chain CS adder with hybrid formatted inputs, or (4) as a 16×4 multiplier, (5) or as part of a larger 16×16 multiplication process. Apart from the aforementioned operation modes, the enhanced pipelined stage (pipeline stage 0 in Fig. 6) can perform also (1) two independent CS-additions/subtractions, or (2) one independent single addition/subtraction and one 4 operand chain addition etc.

The proposed architecture can be reconfigured in a cycle by cycle basis by writing configuration words to the configuration context register. The context register drives all the multiplexors allocated inside or outside of the RAU.

The format of the configuration word is depicted in Fig. 9. It is organized in groups of control bits, starting from the bits that control the configuration of RAU's pipeline stage 0, continuing with the bits which control the other two pipeline stages (pipeline stage 3 needs no configuration) and ending with the control bits dedicated to the remaining multiplexors of the kernel architecture. For the architecture solution analyzed in Section 4, the format of the configuration groups for pipeline stages 1 and 2 are the same. The modular organization of the configuration word enables easy parametrization of the reconfigurable kernel architecture in order to synthesize architectural prototypes with various degrees of configurability (Section 6.3).

The heterogeneity between the pipeline stages is reflected into the configuration word. Thus, the configuration group of pipeline stage 0, which delivers higher configurability, is larger (25 bits long) in comparison with the configuration group of pipeline stage 1 (15 bits long). The configurability of each pipeline stage affects the configuration group of the non-RAU components too. This happens because the number of outputs ports per pipeline stage, which form the set of inputs of the non-RAU multiplexors, alternate according to the configuration capabilities of each pipeline stage.

We have considered the following naming convention rules, in order to make readable the configuration's word format (Fig. 9). Inside each configuration group, the name of the subwords which control the operation mode, starts with the sequence "M0", "M1" or "M2" according to their functionality, and is followed by the identifier "SEG" for segment and the pipeline's segment number. The remaining subwords inside each configuration group control the multiplexors allocated in the reconfigurable kernel architecture. Their name starts with the identifier "SEL" for selection, followed by the name of the component which each subword refers to (i.e., SEL_RF2: subword controlling the input selection of the second register in the register file).

Each configuration group in Fig. 9, targeting the reconfigurable pipeline stages, starts with the configuration bits controlling the UCs' functionality. The bits ($M0\_SEG_i$, $M1\_SEG_i$, $i \in \{1,2\}$) control each UC's function, which is either addition, subtraction or multiplication. The configuration bit $M2\_SEG_i$ controls the intra-RAU multiplexors which pass either an external input data or the RAU's internal propagated data (Fig. 6). For pipeline stage 0, the configuration subwords ($M0\_SEG0\_Line(i)$, $M1\_SEG0\_Line(i)$, $i \in \{0,1,2,3\}$) configure each addition line separately. Thus they enable the pipeline stage to perform complex chained operations (i.e., add-sub-add, two parallel add-sub sub-add etc.). Table 3 shows the operation modes of each line in pipeline stage 0, according to the values of $M0\_SEG0\_Line(i)$, $M1\_SEG0\_Line(i)$ control signals. Pipeline segments 1 and 2 do not support per addition configurability and they can be configured either as 4 chained adders/subtractors or 16×4 multipliers. This is the reason why the configuration subwords of pipeline stages 1 and 2 include only 1-bit for the signals ($M0\_SEG_i$, $M1\_SEG_i$).

The remaining subwords, inside each configuration group of pipeline segments, select the input operands for each pipeline stage. Given that the input data can be in the CS arithmetic format, the selection multiplexors pass 32-bit wide words (2×16-bit), concatenating together the sum and the carry bits of a CS number. Each CS formatted input refers to two 16-bit input ports of the RAU. In case that a conventional binary formatted number is about to be processed in the RAU, the carry's 16-bits are passed as zeros. In the configuration group of pipeline segment 0, there are four configuration subwords of selection bits ($SEL\_IN_i\_IN_{i+1}$), since pipeline stage 0 incorporates up to 8 binary inputs($= 4$ CS inputs). Each of these configuration subwords consists of 4-bit, in order to select between the sixteen CS formatted inputs. The other two reconfigurable pipeline stages(= pipeline

Table 2
Operation level reconfigurability of each pipeline stage in Fig. 5.

Operation mode	Pipeline stage 0	Pipeline stage 1	Pipeline stage 2	Pipeline stage 3
Chain-adder of up to 6 binary or up to 3 CS formatted integers	√	√	√	–
Chain subtractor of up to 6 binary formatted integers	√	√	√	–
Chain subtractor of up to 6 binary or up to 3 CS formatted integers	√	√	√	–
Chain-adder with both CS and binary formatted integers	√	√	√	–
Chain subtractor with both CS and binary formatted integers	√	√	√	–
Two independent chain-adders each of up to 4 binary or up to 2 CS formatted integers	√	–	–	–
Two independent chain subtractors each of up to 4 binary or up to 2 CS formatted integers	√	–	–	–
One chain-adder and one chain subtractor each of up to 4 binary or up to 2 CS formatted integers	√	–	–	–
Two independent add–sub or sub–add operations each of up to 4 binary formatted integers	√	–	–	–
16×4 multiplier	√	√	√	√

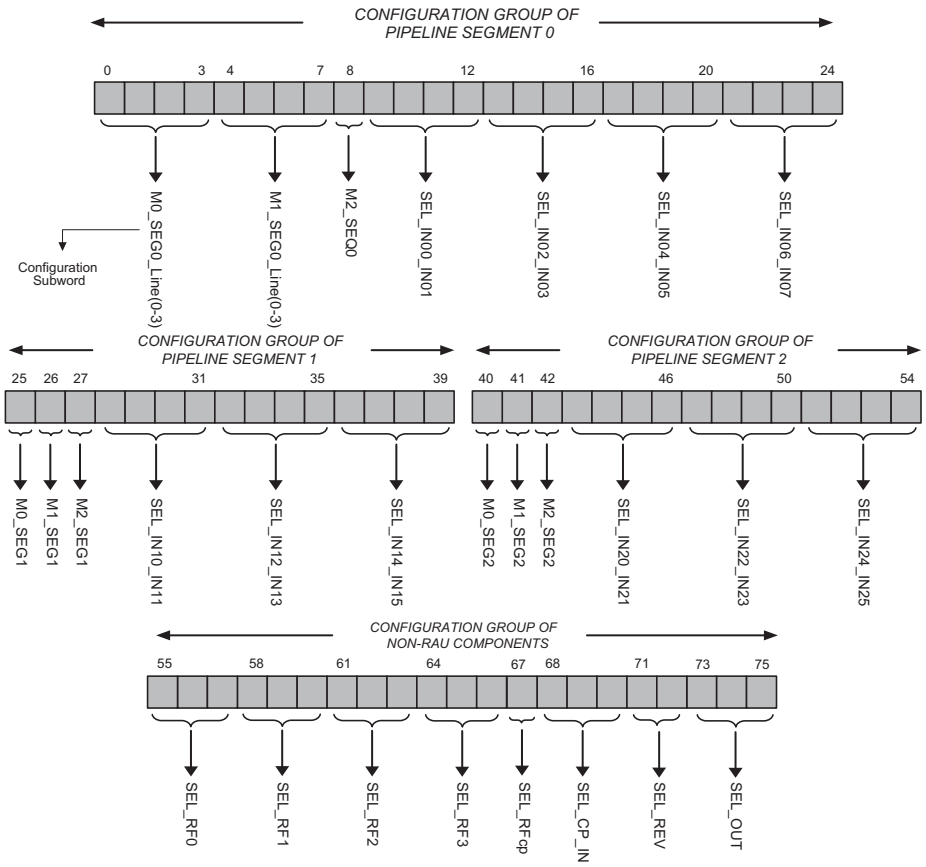

Fig. 9. The format of the configuration word.

Table 3
Operation mode of line $i \in \{0, 1, 2, 3\}$ of pipelined stage 0.

Operation	*M0_SEG0_Line*(*i*)	*M1_SEG0_Line*(*i*)
Addition	0	0
Subtraction	1	0
Multiplication	0	1
Do not Care	1	1

stage 1,2) incorporates 6 binary inputs. Thus, three configuration subwords of selection bits exist in their configuration group. The non-configurable pipeline stage does not contribute at all to the configuration word. It accepts the horizontal multiplicand from the previous pipeline stage and the vertical multiplicand from the synchronization mechanism (Section 4, Fig. 8).

The last configuration group controls all the non-RAU components. Each register in the register file is controlled by three control bits (SEL_RF_i) which select between six inputs. The CS number that is going to be converted is selected by the *SEL_CP_IN* subword of the configuration word. One bit (SEL_RF_{CP}) controls if the converted data are going to be stored at one 16-bit register for future use or if they are going to be fed into the RAU just after the arithmetic conversion. The 16-bit operand that counts for reversing its bit order is selected by the *SEL_REV* configuration bits. Finally, the output of the reconfigurable architecture results from the *SEL_OUT* bits of the configuration word.

The configuration word of the proposed reconfigurable architecture is up to 76-bits wide which resembles the instruction length of current VLIW processors [29,30]. The configuration word can of course be encoded but this would require a decoder unit to be embedded into the kernel architecture augmenting the critical

Table 4
Qualitative comparison between coarse-grain reconfigurable cells.

Kernel	Functions	Allocated resources	Arithmetic format
RAU	Chained/single addition, chained/single subtraction, multiplication	Mux-enhanced multiplier	Carry save, unsigned binary
Cell in [13]	Single ALU operation or multiplication	One ALU unit and one multiplier	Unsigned binary
Cell in [12]	Chained mult-ALU or mult–mult operations	Four ALUs and four multipliers	Unsigned binary
[15]	Single addition or multiplication	One Mux-enhanced multiplier	Unsigned binary
[16]	Multiplication	One decomposed multiplier with input duplication network	Unsigned binary, 2's complement
[17]	Single addition or multiplication	One decomposed multiplier/adder with input duplication network	Signed binary, 2's complement
[18]	Multiplication	Mux-enhanced multipliers shifters, adders	Unsigned binary
[19]	Multiplication	One partitioned Mux-based multiplier	2's complement

path and the overall area of the implementation. For these reasons, we preferred to fully expose the control of the kernel architecture delivering the original configuration word to the context register.

6. Experimental results

In this section, we demonstrate the effectiveness of our architecture and we explore its hardware characteristics. The evaluation of our architecture was made in a qualitative and in a quantitative manner. We performed three sets of quantitative comparisons. The first set evaluates the reconfigurable kernel architecture presented in Section 4 in comparison to non reconfigurable arithmetic circuits and other coarse-grained reconfigurable arithmetic architectures recently presented in the literature. For comparison reasons, the proposed architecture was coded in Verilog-HDL and was mapped onto a classical FPGA device. The functional verification of the proposed architecture was made using the ModelSim simulation environment [33]. Note that the proposed coarse-grained architecture does not target the FPGA's fine-grained implementations. It was mapped onto a FPGA device only for straightforward comparison with the other coarse-grained architectures which have been evaluated using the same FPGA device. The second set of quantitative experiments took place considering the ASIC design space. Alternative implementation scenarios based on the proposed architecture abstract model were explored in terms of the configurability's degree of each reconfigurable pipelined stage. The scalable behavior of the proposed architecture along with its hardware characteristics, in a quantitative manner, are reported. For this set of quantitative experiments, parameterized reconfigurable kernel templates have been synthesized using the Synopsys Design Compiler [32] and the TSMC 0.13 um technology library [34], following a standard cell design methodology. The third set of experiments evaluates the dominant components of the proposed architecture based on the implementation technology (FPGA or ASIC) and explores different chained addition schemes that can be incorporated into the reconfigurable pipeline stages.

6.1. Qualitative comparisons

The proposed RAU was qualitatively compared with the coarse-grained solutions presented in Section 2. For the sake of judicial comparisons, in case of array-based architectures such as [13,12], we considered only their basic reconfigurable cells and not the whole array of the architecture. The comparative results are reported in Table 4. The second column of Table 4 reports the operations that can be mapped onto each referenced architecture, while the third column shows the basic allocated resources which form each architecture. RAU based architectures enable the highest degree of different mapped operations among all the others reconfigurable architectures, except for the case of the cell in [12], which presents the highest degree among all. However, the cell in [12] allocates a large number of computational resources. Our architecture enables these functions with only one enhanced multiplier resource allocated, with significantly larger clock frequencies (Section 6.2) than the cell in [12]. The last column of Table 4 shows that the proposed architecture is the only one which manipulates both unsigned binary and CS formatted data (for fast chained add/sub computations) permitting also mixed computations between them.

6.2. Quantitative comparisons with reconfigurable and dedicated architectures mapped onto FPGA devices

The proposed architecture was mapped onto a Xilinx Virtex-II xc2v3000 device [10] and it was compared with (1) non reconfigurable MAC units and multipliers and (2) other reconfigurable architectures [17], [18] presented in Section 2, all implemented in the same Virtex-II FPGA device. The experimental procedure followed in all cases is based on the HDL-to-bitstream toolflow provided by Xilinx [10]. We performed measurements which are provided in Tables 5 and 6. The measurements for our architecture are based on the real implementation data, after the completion of the place and route process onto the Virtex-II xc2v3000 FPGA device.

Table 5 reports comparison results among the instantiated reconfigurable kernel architecture (Section 4), the coarse-grained reconfigurable MAC unit presented in [17] and two versions of a non reconfigurable MAC unit. The first version of the MAC, MAC_{v1}, makes use of a CS array multiplier, which was mapped onto the programmable slices of the device. The second version of the MAC, MAC_{v2}, is based on the dedicated embedded multiplier, which is available into the Virtex-II FPGA device as a hard-macro. The addition component for both non reconfigurable MACs is a 16-bit carry-look-ahead adder [22].

In comparison with the reconfigurable MAC in [17], our architecture delivers better results in almost all the cases. The hardware complexity of the proposed architecture is 46.9% smaller in terms of number of slices. Also, it is able to operate with a clock frequency of 55.5 MHz, which outperforms the maximum frequency of the architecture presented in [17] in a size of magnitude close to $\times 4$. Considering the MOPS metric, the proposed architecture delivers a gain factor of 32.75%. The value of MOPS is defined by the number of operations multiplied by the clock frequency divided by the operation latency (= number of cycles), as in [17]. The power consumption in terms of mW/MHz is 25.2% lower. However, in terms of MOPS/mW the architecture in [17] has a better value. This is explained by the fact that the actual power of our architecture, in termsof mW, is much higher than the actual power of [17], since the proposed architecture runs at higher clock frequency and it is based in a high pipelined

Table 5
Quantitative comparison between the proposed architecture, the architecture in [17] and non-reconfigurable MACs mapped onto virtex-II FPGA.

Arch.	Area (# slices)	Clock freq. (MHz)	Power (mW/MHz)	No. of operands	Op. density (#op./Area)	MOPS	MOPS/mW
√ Proposed	3187	55.5	7.63	20	6.27	333	0.79
√ Ref. [17]	6013	13.996	10.2	32	5.32	223.94	1.56
Gains (%)	46.9	296.5	25.2	–	17.8	48.70	–
√ MAC_{v1}	374	52.26	6.58	2	5.3	104.52	0.30
Gains (%)	–	6.2	–	900	18.3	218.6	163.33
√ MAC_{v2}	76	90	3.82	2	N/A	180	0.52
Gains (%)	–	–	–	900	N/A	85	51.92

Table 6
Quantitative comparison between reconfigurable and non-reconfigurable multiplication units mapped onto virtex-II FPGA.

Reconf. modules	Area (# slices)	Min. clock delay (ns)	Power (mW/MHz)	MOPS	MOPS/mW
16 × 16 RAU	1361	9.98	7.17	250.2	0.35
16 × 16 in [17]	2552	18.5	N/A	288	N/A
16 × 17 in [18]	729	28.7	13.4	69.7	0.15
16 × 16 CS-mult	315	19	6.45	52.26	0.15

structure. We remind though, that the proposed architecture targets the ASIC domain, which does not suffer from the high quiescent power dissipation like the Xilinx Virtex-II FPGA devices. Further information about the power consumption of the proposed architecture implemented in the ASIC domain is given in Section 6.3. We have also compared the number of input operands that the two architectures can operate on, for the case that the architecture in [17] is configured to perform 16-bit multiplications. The proposed architecture operates on twenty 16-bit formatted operands while the architecture of [17] is able to operate on thirty two. However, in terms of operand's density, specified as *#operands/Area*, our architecture delivers a higher rate of 17.8%.

We also compared the proposed architecture with the two non reconfigurable MACs. As expected, the non reconfigurable architectures are more efficient in area than the proposed reconfigurable architecture. We have to mention that the 76 slices reported in the MAC_{v2} refers only to the logic mapped onto the reconfigurable part and do not include the area of the hard-macro multiplier. This is the reason why we excluded the *#operands/Area* result from Table 5, in case of MAC_{v2}. The proposed reconfigurable architecture delivers gains up to 18.3% in operands density, 218.6% in MOPS and 163.33% in MOPS/mW which makes it an energy efficient solution compared with the MAC_{v1}. Comparing to the MAC_{v2} implementation, our architecture provides 85% and 51.92% gains in case of MOPS and MOPS/mW metric, respectively.

We have also compared in Table 6 the RAU component of the kernel architecture against reconfigurable multiplication units presented in other approaches. The RAU, the 16 × 16 multiplication modules presented in [17,18] and a 16 × 16 CS-array multiplier were mapped onto the Virtex-II xc2v3000 FPGA device. Given that RAU supports multiplication operations along with conventional and chained addition/subtraction ones, the MOPS value resulted as the mean value of the MOPS when the RAU is configured to perform multiplication and when the RAU is configured to perform chain addition or subtraction operations. RAU is able to operate approximately ×2 and ×3 faster that the multiplication units in [17] and in [18], respectively. Gains up to 256% in MOPS value and 133% in MOPS/mW are reported comparing to [18]. In general, RAU outperforms the other two reconfigurable solutions except for the area coverage compared with the multiplication unit in [18] and for the MOPS metric compared with [17]. However, the solution in [18] has been specially designed for Xilinx Virtex-II FPGA devices while the solution in [17] supports only bitwidth based reconfigurability. In comparison with the NR CS-array multiplier, RAU is able to operate ×2 faster and delivers gains up to 198% and 133% considering the MOPS and MOPS/mW metrics, respectively.

The results reported in Table 6 considered RAU as well as the other multiplication units, as stand-alone components mapped onto the Virtex-II FPGA. When RAU is measured as part of the overall reconfigurable kernel architecture (clock frequency 55.5 MHz), the power consumption of RAU is 5.93 mW/MHz while the MOPS value of RAU becomes 138.75.

From the comparisons taken place in Sections 6.1 and 6.2, it can be stated that, the proposed architecture forms an efficient solution which combines the advantages of both the array-based reconfigurable architectures and the reconfigurable multiplier/MAC architectures. It offers fast implementation of arithmetic behaviors along with high reconfiguration capabilities at the operation level without consuming large portions of hardware computational resources, like the most of the array-based architectures. Also, it is not strictly application specific in one arithmetic behavior like the bitwidth-based reconfigurable multipliers/MACs. For that reasons, the proposed architecture can be a promising solution for applications which require high performance along with operation level flexibility, like those found in the DSP domain.

6.3. *Reconfigurability based quantitative exploration of the proposed architectural templates in the ASIC design space*

This set of experiments concerns the exploration of alterative implementation scenarios for the abstract model of the proposed architecture, considering the configurability degree of each pipeline stage. The scaling behavior of the proposed architecture model among different configurability's degrees is reported and its hardware characteristics are evaluated quantitatively. Synthesis of the reconfigurable architecture according to the standard cell design methodology was performed. Eight different architectural templates of the proposed architecture varying in their dynamical reconfigurable characteristics were modeled, synthesized and evaluated. The architectural templates were modeled

using the Verilog hardware description language and were synthesized using the Synopsys Design Compiler [32] with the TSMC 0.13 um ASIC CMOS technology library [34].

We adopted a hierarchical design procedure, starting from the bottom hierarchy level and combining structurally the Verilog modules towards the top level hierarchies. All the templates were synthesized under the timing constraint of 300 MHz clock frequency and mapped onto the standard cells provided by the TSMC 0.13 um library. We consider typical operating conditions and automatic wire load model selection, except for the case of the RAU's input multiplexors (the interconnection network in Fig. 5) where the aggressive wire model parameter was set to optimize the imposed delays. For further optimization of the global interconnection scheme, the top level design module was flattened during the synthesis procedure.

The eight templates of the reconfigurable kernel architecture derived by changing the reconfiguration capability of each pipeline stage found in RAU. Each pipeline stage can be configured either as a complex reconfigurable (CR) stage like the pipeline stage 0 in Fig. 6 or as a single reconfigurable (SR) stage like the pipeline stage 1 in Fig. 6, or as a non reconfigurable (NR) stage like the last pipeline stage of the RAU. Given the stable interconnection scheme inside the RAU's pipeline stages, the area overheads of the SR stages over the NR stages are imposed only from the extra allocated hardware of the UC cells, in respect to the conventional multiplier's cells (Section 3.3). The area overheads of the CR stages over the SR stages are imposed by the intermediate multiplexer line and the higher wiring due to the extra two output ports allocated.

The overall configurability of the RAU has a great impact on the whole architecture and especially on the number and the width of the allocated steering logic. Table 7 shows the eight architectural templates (*ARCH*0-*ARCH*7) characterized by the RAU's reconfiguration capability. The templates have been sorted in a descending order based on their Remanence metric [31]. The remanence characterizes the dynamical character of the reconfigurable architecture. The higher the remanence's value is, the more configurable the underlying hardware architecture becomes. For a reconfigurable architecture composed of N_{PE} reconfigurable processing elements, running at the clock frequency F_e and configuring N_c processing units at each reconfiguration cycle of frequency F_c, the Remanence is defined by the following formula:

$$Remanence = \frac{N_{PE} \cdot F_e}{N_c \cdot F_c} \tag{7}$$

For all the architectural templates $N_c = 4$ and $F_e = F_c =$ 300 MHz. We include the last NR pipeline stage in the N_c value given that it can be configured either as CR or SR pipeline stage too. The N_{PE} depends on the configuration of each pipeline stage and it is different in each architectural template. *ARCH*2 is the instantiated architectural template which we analyzed in Section 4.

Table 7
Characterization of the architectural templates.

Architecture template	Pipeline stage 0	Pipeline stage 1	Pipeline stage 2	Pipeline stage 3	Remanence
ARCH0	CR	CR	CR	NR	1.5
ARCH1	CR	CR	SR	NR	1.25
ARCH2	CR	SR	SR	NR	1.0
ARCH3	CR	CR	NR	NR	1.0
ARCH4	SR	SR	SR	NR	1.0
ARCH5	CR	SR	NR	NR	0.75
ARCH6	CR	NR	NR	NR	0.5
ARCH7	SR	NR	NR	NR	0.25

CR: 1 chained add/sub, or two smaller chained adds/subs in parallel, or 16×4 mult. SR: one chained add/sub, or 16×4 mult. NR: 16×4 mult.

In Fig. 10a the hardware area (um^2) of the synthesized architectural templates is depicted. In Fig. 10b the power consumption (mW) for each template is reported. The tables under the diagrams shows the active area/power estimates for each architectural template. As expected, the area scales down moving towards the architectural templates with smaller remanence values. The occupied hardware area is affected mainly by the input and output ports of the RAU which contribute to more feedback wires and to larger input selection multiplexors. Fig. 10a depicts that, the area complexity of the reconfigurable architecture scales almost linear with respect to the incorporated reconfigurability. The same scaling behavior is observed in the power consumption diagram as well. This feature of linear scaling gives the advantage to the designer to select among different architectural solutions without worrying about unexpected area/power overheads. However, it has to be noted that the number of pipeline stages into the RAU has been restricted to four. For RAUs with more than four pipeline stages, the area/power scaling factor is expected to approximate an exponential growth as the number of RAU's available pipelines augments.

Driven by the fact that the RAU is the dominant computational component in the proposed architectural template, we provide the area-configurability and the power-configurability diagrams for the different RAU architectural templates (Fig. 11). As in the case of the overall reconfigurable architecture, the area coverage of RAU scales linearly among the different RAU templates. The gradient though in Fig. 11a is much lower than in Fig. 10a, since the measurements concerns only the RAU component and they did not include the area of the input multiplexors.

The interesting point of the measurements reported in Fig. 11 is that for the RAU modules with the same remanence values (*ARCH*2, *ARCH*3, *ARCH*4) there are significant differences in the allocated hardware area and the dissipated power. For example, *ARCH*3 delivers smaller area occupation and power consumption than *ARCH*2 and *ARCH*4. We can infer, that for a given remanence value it is preferable to use CR pipeline stages along with more NR ones, rather than many SR pipeline stages. Especially for applications that do not need many chained additions/subtractions in parallel, the aforementioned feature of the RAU can generate solutions with low area and power overheads without performance degradation.

6.4. Evaluation of implementation targets and design alternatives

In Sections 6.2 and 6.3, the proposed reconfigurable architecture was mapped onto two different implementation technologies, namely FPGA and ASIC, respectively. In this subsection, the performance differences between the two implementations are quantitatively evaluated. We experimentally prove our previous statement that the proposed architecture targets the ASIC technology rather the FPGA's one. Based on this result, we further investigate, in the ASIC domain, alternative chain-addition schemes for the internal structure of the reconfigurable pipeline stages.

In this set of experiments we synthesized separately (1) the SR pipeline stage and (2) the multiplexer based interconnection network (Fig. 5). The two components were coded in technology independent Verilog-HDL using structural RTL coding style. The HDL descriptions were mapped onto (1) the Xilinx Virtex-II xc2v3000 FPGA device [10] (fabrication technology parameters: 0.15 um 8-layer metal process with 0.12 um high-speed transistors) and (2) onto the standard cells provided by the TSMC 0.13 um ASIC library [34].

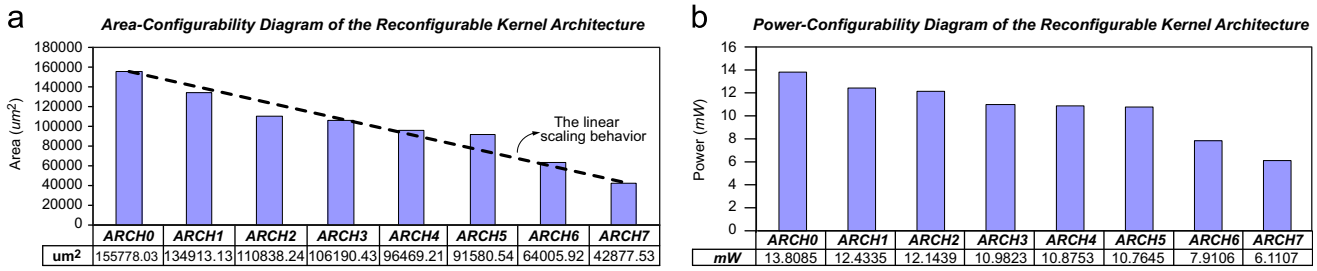

Fig. 10. The reconfigurable kernel's architecture: (a) area-configurability diagram and (b) power-configurability diagram.

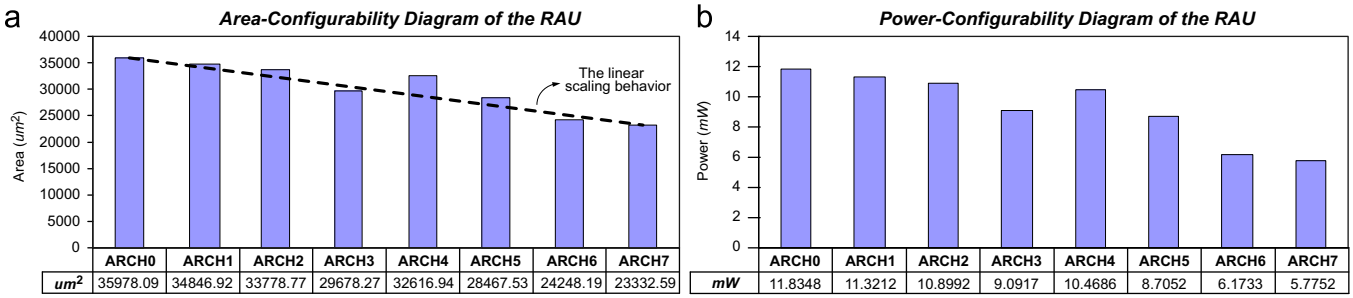

Fig. 11. The RAU's: (a) area-configurability diagram and (b) the power-configurability diagram.

Table 8
Evaluation of the RAU mapped onto FPGA and ASIC libraries.

Target technology	Optimization criterion	Actual area	Equiv. gate count	Critical path (ns)	Power (mW/MHz)
Virtex-II FPGA	Speed	201 slices	2925	11.1	3.66
	Area	201 slices	2925	11.1	3.66
TSMC 0.13 um	Speed–area	13486.72 um^2	1134.29	2.0	0.025

Table 9
Evaluation of the multiplexer-based interconnection network mapped onto FPGA and ASIC libraries.

Target technology	Optimization criterion	Actual area	Equiv. gate count	Critical path (ns)	Power (mW/MHz)
Virtex-II FPGA	Speed	1220 slices	17400	3.08	1.02
	Area	1206 slices	17820	2.85	0.94
TSMC 0.13 um	Speed–area	43474.64 um^2	3657	0.5	0.01

Tables 8 and 9 report the comparative results of the two implementation technologies. The second column of Tables 8 and 9 refers to the optimization criterion which guided the synthesis procedure. While Synopsys Design Compiler [32] enables synthesis taking into account both optimization criterions simultaneously, Xilinx tool-flow permits only one optimization criterion to guide the synthesis procedure. We provide FPGA results separately for speed constrained synthesis and area constrained synthesis. The equivalent gate count metric is provided for a straightforward comparison of the area efficiency between the two implementation targets.

At first, some interesting results are reported comparing only the two FPGA implementations. The area, critical path and power dissipation of the SR pipeline stage (Table 8) are the same for both optimization criterions. The UCs' in every pipeline stage have been designed at the gate level of abstraction (Section 3.3) and the optimization criterion does not affect the final netlist. In case of the multiplexer-based interconnection network (Table 9) the area optimized FPGA design delivers smaller area, critical path delay and power dissipation than the speed optimized. Although that the actual area (#slices) of the FPGA device is smaller in the area optimized netlist of the interconnection network, the equivalent gate count has a larger value than the speed optimized one. The area optimization criterion enables more compact placement and routing onto the FPGA device than the speed optimization criterion. More compact placement and routing deliver smaller routing tracks which are the reason that the critical path delay is smaller in the area optimized than in the speed optimized implementation.

The ASIC implementations outperform the FPGA ones in all cases, as expected. The ASIC implementation of SR pipeline stage delivers $\times$1.58 gain in area complexity, $\times$4.55 gain in critical delay and $\times$145 gain in power dissipation (Table 8), when it is compared with the FPGA one. Additionally, the ASIC implementation of the multiplexer based interconnection network (Table 8) delivers $\times$3.75 gain in area coverage, $\times$4.7 gain in critical path's delay and

Table 10
Comparison between different chain addition schemes for the intra-pipeline stage structure.

Addition schemes	Area (um^2)	Critical path (ns)	Power (mW/MHz)
Chained carry save (UC_{OPT})	13486.72	2.0	0.025
Chained carry save (UC_{SUBOPT})	13942.72	2.0	0.030
Chained carry lookahead	15283.38	2.9	0.049
Chained Manchester	13815.12	4.0	0.056
Chained ripple carry	14038.32	4.4	0.066

$\times$93 gain in power consumption, in comparison to the best FPGA implementation (area optimized). As stated in Section 6.2, FPGA implementation was considered only for straightforward comparisons with previously published reconfigurable arithmetic architectures.

Table 10 reports exploration results, based on various chain-addition schemes, for the internal structure of a SR pipeline stage. Specifically, we consider the (1) CS, (2) carry look-ahead, (3) Manchester carry-chain and (4) carry ripple single addition schemes [22]. Based on these schemes, chained adders with four single addition units were formed and they were enhanced in order to execute multiplication and chained subtraction as well. Thus, various SR pipeline stages were generated based on the different chain-addition schemes and were synthesized using the Synopsys Design Compiler [32] and the standard cell ASIC technology library TSMC 0.13 um [34]. Specifically, we iterate the synthesis procedure for each of the aforementioned SR schemes, with a timing constraint interval of 0.1 ns, until no timing violations were reported.

We considered five variants of the RAU's SR pipeline stage, reported in the first column of Table 10. The two CS variants refer to the same internal structure altering the UC implementation strategy (Table 1). The UC_{OPT} based implementation delivers smaller area and power dissipation than the UC_{SUBOPT} implementation. In general, the proposed UC_{OPT} based CS chained addition delivers the smallest critical path's delay and power consumption among all other solutions. It is up to 0.9 ns faster, it dissipates approximately 50% less power and it occupies 11.75% less area than the chained carry look-ahead scheme. Also, the proposed chained addition scheme occupies less area and is approximately $\times$2 faster and power efficient than the chained Manchester implementation. Comparing with the carry ripple based implementation, the proposed CS chained addition scheme delivers even higher gains in critical delay and power dissipation.

7. Conclusion

In this paper, we have presented a design methodology for coarse-grained reconfigurable datapaths. Our main goal is the insertion of flexibility into the computational resources of custom arithmetic datapaths in order to provide operation level reconfigurability. Based on the specific design methodology, a novel coarse-grained reconfigurable architecture has been introduced. We have qualitatively and quantitatively evaluated and compared the proposed reconfigurable solution with other previously published reconfigurable and non-reconfigurable datapaths. Comparisons have shown the advantageous features of the proposed architecture considering its area, time and power efficiency and its reconfiguration capability. Future work includes the development of a comprehensive synthesis methodology and the corresponding tools in order to enable automated code mapping of DSP specifications onto the proposed reconfigurable architecture.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments and suggestions and Professor Dimitrios Soudris for the long and constructive discussions during the preparation of the manuscript. This research was partially supported by the E.C founded program MOSART IST-215244, website: http://www.mosart-project.org.

References

[1] K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software, ACM Computing Surveys 34 (2) (2002) 171–210.
[2] S. Vassiliadis, D. Soudris, Fine- and Coarse-Grain Reconfigurable Computing, Springer, Berlin, 2007, pp. 3–149.
[3] H. Singh, M. Lee, F.K.G. Lu, N. Bagherzadeh, E. Filho, Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications, IEEE Transactions on Computers 49 (5) (2000) 465.
[4] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, J. Rabaey, Design methodology of a low-energy reconfigurable single-chip DSP system, Journal of VLSI Signal Processing 28 (1–2) (2001) 47–61.
[5] S. Wallner, A configurable system-on-chip architecture for embedded and real-time applications: concepts, design and realization, Elsevier Journal of Systems Architecture 51 (6–7) (2005) 350.
[6] A. Olugbon, S. Khawam, T. Arslan, I. Nousias, I. Lindsay, An AMBA AHB-based reconfigurable SoC architecture using multiplicity of dedicated flyby DMA blocks, in: Proceedings of ASP-DAC, 2005, pp. 1256–1259.
[7] S. Hauck, T. Fry, M. Hosler, J. Kao, The chimaera reconfigurable functional unit, in: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1997, pp. 87–96.
[8] Xilinx Corporation, Xilinx Spartan Series, ⟨www.xilinx.com⟩.
[9] R. Kastner, S. Ogrenci-Memik, E. Bozorgzadeh, M. Sarrafzadeh, Instruction generation for hybrid reconfigurable systems, ACM Transactions on Design Automation of Electronic Systems 7 (4) (2002) 605–627.
[10] Xilinx Corporation, Xilinx Virtex Series, ⟨www.xilinx.com⟩.
[11] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabaey, Performance optimization using template mapping for datapath-intensive high-level synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15 (2) (1996) 877–888.
[12] M. Galanis, G. Theodoridis, S. Tragoudas, C. Goutis, A high performance datapath for synthesizing DSP kernels, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (6) (2006) 1154–1163.
[13] M. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. Kurdahi, Design and implementation of the morphosys reconfigurable computing processor, Journal of VLSI Signal Processing Systems 24 (2000) 147–164.
[14] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, ADRES: an architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix, in: Proceedings of the International Conference on Field Programmable Logic and Applications, 2003, pp. 61–70.
[15] S. Chiricescu, M. Schuette, R. Glinton, H. Schmit, Morphable multipliers, in: Proceedings of the International Conference on Field Programmable Logic and Applications, 2002, pp. 647–656.
[16] R. Lin, Reconfigurable parallel inner product processor architectures, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9 (2) (2001) 261–272.
[17] K. Tatas, G. Koutroumpezis, D. Soudris, A. Thanailakis, Architecture design of a coarse-grain reconfigurable multiply-accumulate unit for data-intensive applications, Integration the VLSI Journal 40 (2) (2007) 74–93.
[18] P. Corsonello, S. Perri, M. Iachino, G. Cocorullo, Variable precision multipliers for FPGA-based reconfigurable computing systems, in: Proceedings of the FPL, Lecture Notes in Computer Science, Springer, Berlin, 2003, pp. 661–669.
[19] O. Al-Khaleel, C. Papachristou, F. Wolff, K. Pekmestzi, A large scale adaptable multiplier for cryptographic applications, in: Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems, AHS, 2006, 2006, pp. 477–484.
[20] V. Van Dongen, P. Quinton, Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays, in: Proceedings of the IEEE International Conference on Systolic Arrays, 1988, pp. 473–482.
[21] M. Manjunathaiah, G.M. Megson, S. Rajopadhy, T. Risset, Uniformization of affine dependence programs for parallel embedded system design, in: Proceedings of IEEE International Conference on Parallel Processing (ICPP '01), 2001, pp. 205–213.
[22] N. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, third ed, Addison-Wesley, Reading, MA, 2005.
[23] L. Chiou, S. Bhunia, K. Roy, Synthesis of application-specific highly efficient multi-mode cores for embedded systems, ACM Transactions on Embedded Computing Systems (TECS) 4 (1) (2005) 168–188.
[24] W. Zhao, C. Papachristou, Synthesis of reusable DSP cores based on multiple behaviours, in: Proceedings of ICCAD, 1996, pp. 103–108.
[25] R. Brayton, et al., Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston, MA, 1984.

[26] T. Sasao, Input variable assignment and output phase optimization of PLA's, IEEE Transactions of Computers 33 (10) (1984) 879–894.
[27] F. Elguibaly, Overflow handling in inner-product processors, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 47 (10) (2000).
[28] S. Perri, P. Corsonello, G. Cocorullo, A high-speed energy-efficient 64-bit reconfigurable binary adder, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 11 (5) (2003) 939–943.
[29] Texas Instruments Inc., TMS320C6000 Family: High Performance DSP Platform, available at: ⟨http://www.ti.com⟩.
[30] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, F. Homewood, Lx: a technology platform for customizable VLIW embedded processing, in: Proceedings of the 27th Annual international Symposium on Computer Architecture, ISCA '00, 2000, pp. 203–213.
[31] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, G. Cambon, Metrics for reconfigurable architectures characterization: remanence and scalability, in: Proceedings of the Parallel and Distributed Processing Symposium, 2003, April 2003, pp. 22–26.
[32] Synopsys Inc., Design Compiler version 2004, ⟨www.synopsys.com⟩.
[33] ModelSim, ⟨http://www.model.com⟩.
[34] Artisan Components, TSMC 0.13 Library Databook.

Sotirios Xydis received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2005. Currently, he is working towards his Ph.D. at National Technical University of Athens (NTUA) in the Department of Electrical and Computer Engineering. His research interests include reconfigurable architectures, design and optimization of arithmetic VLSI circuits and high-level synthesis algorithms. He is an IEEE student member and has published over 10 technical and research papers in international journals and conferences. He is the recipient of the best paper award in Reconfigurable Systems category of NASA/ESA/IEEE International Conference on Adaptive Hardware and Systems 2007 (AHS 2007).

George Economakos received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1992. He received the Ph.D. Degree in Electrical and Computer Engineering, from the National Technical University of Athens in 1999. He is currently working as Lecturer of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research interests include design automation, high-level synthesis, electronic system level design, reconfigurable computing and design for low power. He has published more than 80 papers in international journals and conferences and served as a reviewer in most of them, being a member of the program committee four times. He was investigator in numerous research projects funded from the Greek Government and Industry as well as the European Commission. He is a member of the ACM, IEEE, EUROMICRO and ECSI and has served as a member in more than 10 standardization groups in the field of design automation.

Kiamal Pekmestzi received his Diploma in Electrical Engineering from the National Technical University of Athens (1975). From 1975 to 1981 he was a research fellow in the Electronics Department of the Nuclear Research Center "Demokritos". He received his Ph.D. in Electrical Engineering from the University of Patras (1981). From 1983 to 1985 he was a Professor at Higher School of Electronics in Athens. Since 1985 he has been with the National Technical University of Athens, where he is currently a Professor in the Department of Electrical and Computer Engineering. His research interests include efficient implementation of arithmetic operations, design of embedded and microprocessor-based systems, architectures for reconfigurable computing, VLSI implementation of cryptography and digital signal processing algorithms.