INTEGRATION the VLSI journal xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

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

Customizable embedded processor array for multimedia applications

Mehmet Tükel[a,*,1], Arda Yurdakul[b], Berna Örs[a]

[a] Department of Electronics & Communication Engineering, Istanbul Technical University, Istanbul, Maslak TR-34469, Turkey
[b] Department of Computer Engineering, Boğaziçi University, Istanbul, Bebek TR-34342, Turkey

ARTICLE INFO

Keywords:
Customizable Processor Array
Flexible instruction
Image processing hardware
Domain specific computing
Time-to-market

ABSTRACT

We are proposing a Customizable Embedded Processor Array for Multimedia Applications (CPAMA). This architecture can be used as a standalone image/video processing chip in consumer electronics. Its building blocks are all designed to achieve low power and low area, thus it is a good candidate for low cost consumer electronics. Our contribution is, designing a configurable embedded multimedia processor array considering the nature of image/video processing applications. This approach is considered in all the basic blocks of the architecture. Because of its configurable architecture and ability to connect with other devices, it may be used in a large domain of applications. Our architecture is purely implemented with VHDL. It is not dependent on any technology or design software. We have implemented our architecture for different applications on a Xilinx Virtex-5 device and as a number of Application Specific Integrated Circuits (ASIC) by using 90 nm CMOS technology. Experimental case studies show that CPAMA has better or comparable results to the existing similar architectures in terms of performance and energy consumption. Our studies show that throughput of CPAMA is 0.3x–2.4x times better than ADRES. Energy consumption of CPAMA is 31–50% less than ADRES. On the other hand, in one configuration of IDCT application, CPAMA provides 56% less throughput and consumes 55% more energy than ADRES.

1. Introduction

Computing hardware design methodology has evolved significantly over the years. As chips get larger and complexity of each design increases, flexibility and quick time to market in the form of reprogrammable/reconfigurable chips and systems increase in importance [1]. Several Multi Processor System on a Chip (MPSoC) and Coarse-Grained Reconfigurable Architectures (CGRA) have been proposed in recent years [2–4]. Using CGRAs may be preferred for several reasons such as speed, area, power or IP re-usability [3]. Furthermore, comparing to Field Programmable Gate Arrays (FPGA), CGRAs have a shorter reconfiguration time. CGRAs are suitable for systems that require intensive computations. By adjusting the number and structure of processing elements on a CGRA, we can obtain an architecture that meets the requirements of the computation.

Image/video processing is an area where algorithms need intensive computation with high performance. Handling this kind of computation usually requires custom hardware [5]. Considering today's technology, every portable device tends to have a camera, e.g. glasses, watches, smart phones, etc. Each device has its own configuration and requires mostly different features. Designing dedicated hardware for image processing tasks for every device is time consuming and not economically feasible at all. In most devices, image processing tasks are handled using System-on-Chips (SoC) with DSP or GPU cores. If a designer chooses to use commercial SoCs, he/she has to accept what the chip offers, in terms of speed and power dissipation. Those architectures may include redundant parts that might not be used at all. This redundancy leads to extra chip area usage and power dissipation. On the other hand, implementing an image processing task on a CGRA yields efficient results in terms of area, power dissipation, or speed comparing to commercial SoCs [6]. Time-to-market of an image/video processing system, which is implemented on customizable cores like CGRAs, is less than that of a custom Application Specific Integrated Circuit (ASIC) [7]. Besides, it is easy to adopt such systems for later alterations. Consequently, we can say that CGRAs are suitable for image/video processing tasks of low power, low cost consumer electronics.

In this paper, we introduce a Customizable Embedded Processor Array for Multimedia Applications (CPAMA). CPAMA consists of a processor array for intensive computation, and a host processor for control and coordination with other devices. Our configurable architecture is designed by considering the nature and requirements of image processing algorithms:

* Corresponding author.
E-mail addresses: tukel@itu.edu.tr, mehmet.tukel@ankasys.com (M. Tükel), yurdakul@boun.edu.tr (A. Yurdakul), orssi@itu.edu.tr (B. Örs).
[1] Anka Microelectronic Systems.

http://dx.doi.org/10.1016/j.vlsi.2017.09.009
Received 25 January 2017; Received in revised form 6 August 2017; Accepted 29 September 2017

- CPAMA processes a multimedia application in sequences of image blocks. Hence, we design a configurable processor array which concurrently processes all pixels in an image block.
- Each processor of CPAMA can also be configured according to the position of a pixel in an image block depending on the application.

This architecture can be used for domains that require intensive computation such as image/video processing, and scientific computations that can be mapped onto a 2 dimensional (2D) processor array.

This paper is organized as follows: In Section 2 we mention the related architectures in literature and demonstrate the differences with the proposed CPAMA. In Section 3, we explain the basic concepts that we refer in CPAMA design. In Section 4 we present the configurable hardware architecture of CPAMA in details. In Section 5, we present our case study implementations and make comparisons with the existing similar architectures. Finally in Section 6, we make our remarks on the CPAMA architecture and conclude the paper.

2. Related works

Mei et al. [3] proposed a template-based CGRA called Architecture for Dynamically Reconfigurable Embedded System (ADRES). Coarse grained reconfiguration refers to reconfiguration in relatively high level modules, not in logic blocks or in Look Up Tables(LUT) as in an FPGA. A design tool, namely Dynamically Reconfigurable Embedded System Compiler (DRESC) [8], is used for this architecture to generate the design. Propagating data, in other words performing iterations, is implemented in a stream manner. Total performance of the array is strictly related to the effectiveness of scheduling and mapping of the application code onto processing elements, which is handled by DRESC tool. It is known that optimum scheduling in DRESC is an NP-Hard problem. Therefore, the outcome of the scheduler, which is implemented using a heuristic method, is expected to be a sub-optimal solution. Another work related to ADRES [6] suggests that a failure in performance increase despite increasing the size of the array may be caused by a lack of scalability of the scheduling algorithm.

Marshall et al. [9] proposed another CGRA called CHESS, where multimedia applications are taken into account. Despite the reconfiguration word in its definition, most of the features of this array are kept fixed, e.g., number of registers, processors, instructions, etc. The reconfiguration is performed by only changing the program memory. CHESS can be considered as predecessor of ADRES.

Related to CGRAs, data partitioning and instruction scheduling techniques are also studied [10]. The target architecture is a variant of ADRES. Moreover, a recent study [11] focuses on power optimizations on the same target architecture. In these two studies [10,11], the emphasis is not on proposing a new architecture, but on instruction scheduling, data partitioning techniques for a CGRA like ADRES in order to achieve better speed and power consumption results.

Eichel [12] proposed MEP architecture for developing multimedia applications. The architecture consists of a RISC processor and an accompanying VLIW co-processor. The architecture has only instruction level parallelism. The configurable part of the architecture is the VLIW part. It is explained that, the RTL definition of the configurable part is generated based on a customised instruction-set architecture.

Chu et al. [13] proposed a programmable architecture called UniCore. This design is optimised for MPEG4 encoding. The whole architecture is not reconfigurable. It is composed of a 32-bit conventional processor, DSP like units and 4 co-processors. Programmability is achieved by the firmware that runs on the processor and co-processors.

Başsoy et al. [14] proposed an FPGA based customizable processor architecture called SHARF. SHARF has multiple ALU units controlled by the same control unit. ALUs receive instruction addresses from the same bus which is driven by the control unit. ALUs are tightly coupled with the control unit. In this architecture, tightly coupling may cause communication overhead, and moreover may restrict scalability.

Masselos et al. [15] concentrated on low power mapping of multimedia applications on VLIW multimedia processors. They searched methods for mapping tasks on the commercial processors rather than designing their own architectures.

Sanghai and Gentile [16] explored software parallelism in multimedia applications using a dual-core DSP. It is expressed that developing scalable parallel software greatly depends on the efficient use of the interconnect network, memory hierarchy, and the peripheral resources. While designing our CPAMA architecture, we have considered the methods which are proposed for software parallelism in [16].

Rashid et al. [17] proposed implementation of an application specific instruction set processor using a software called LISATek, which is now owned by Synopsys [18]. In this study, the speed-up relies on instruction level parallelism only. A RISC processor provided by LISATek is extended by processor data-path extension using the mechanisms in the tool. In CPAMA, not only instruction level parallelism, but also processor level parallelism is targeted.

Göhringer and Becker [19] proposed a runtime reconfigurable architecture called Runtime Adaptive Multi-Processor System-on-a-Chip (RAMPSoC). Parallel processing elements of the architecture are connected through a Network-on-Chip (NoC) called Star Wheels. This network is composed of groups that may have different number of processing elements. Processing elements of a group can communicate with each other through a switch, and processing elements of different groups can communicate through other larger switches.

Different digital implementations of Cellular Neural Network, which is an analog image processing structure, are proposed in several studies [20–23]. These architectures are capable of filter based image/video processing algorithms.

A configurable video decoder architecture [7] was proposed for mobile terminals. This architecture consists of a conventional application processor accompanied by a co-processor. The co-processor is composed of basic functions (hardware blocks) of H.264 decoder such as loop filter, motion compensation and integer transformation. Communication between the application processor and co-processor is implemented with a bus architecture. This study aims to decrease the time-to-market of a system that requires video decoding, and proposes an adaptable architecture for future standards by making configurable parts.

STP engine [24] is a multi processor accelerator IP, currently provided by Renesas Inc. It is used with a compiler tool called Musketeer [25]. Different stream applications [25,26] are implemented using STP engine. The current version [24] has 256 processing cores with 8-bit word length. STP engine can be considered as a fixed size CGRA with fixed word length.

The literature can be classified into three groups: (1) CGRAs, (2) architectures essentially built for specific applications, and (3) technology/device dependent architectures.

The difference between proposed CPAMA and CGRAs [3,9–11] is that CPAMA consists of fully customizable processors, whereas CGRAs consist of configurable functional units, like Arithmetic Logic Units (ALU). Besides, data are shared by multi-port register files in CGRAs, yet this task is handled through NoC with packets in CPAMA. Some CGRAs [3] have scalability issues. It is hard to comment on performance values in some studies [10,11], because the results are given as normalized values. Last but not least, the problem that needs to be solved on ADRES [8] is stated as a loop expansion problem. Instead in CPAMA, the nature of image processing algorithms is considered as explained in Section 3. Since STP [24] is a hard IP, the number of processing cores and the word length are fixed. On the other hand, in CPAMA, the full architecture is compile-time configurable, including the number of processing cores, word-length, array size and dimensions. Yet, contexts of CPAMA are also generated offline as it is done in STP.

The architectures mentioned in [7,13,20,21,22,22] are essentially

designed for a specific purpose or an application. These architectures are configurable for implementing that specific application, such as filtering, encoding, etc. On the other hand, CPAMA is designed to support several types of image/video processing applications as explained in Section 3.

Some architectures are tailored for a specific device or technology [14,19]. For instance RAMPSoC [19] uses FPGA primitives like reconfiguration ports. This type of primitives make the architecture dependent on even some specific FPGA vendors. The other architecture Sharf [14] is targeting FPGAs as well but not necessarily dependent on them. However, tightly coupling between the controller and the processor elements makes Sharf hardly scalable. Our proposed architecture does not rely on a specific technology or a device, and it is easily scalable. CPAMA is a soft IP. It is written in pure VHDL. Hence, FPGA is only a target platform for our architecture and we do not explicitly use any primitives of an FPGA device. However, when CPAMA is mapped onto an FPGA, the synthesizer maps adders, multipliers, memories, etc. onto primitives of the FPGA. The same design concept also applies for mapping CPAMA to ASIC. Experimental results are given about scalability of our architecture in Section 5 of this paper.

3. Basic concepts of CPAMA

CPAMA is mainly designed to be vastly generic and flexible. In every development cycle of CPAMA, requirements and characters of image processing applications have been considered. Register files of the processors, data-path design, instruction set of the processors, communication among the processors, and FIFO structures are all studied considering the image processing domain. CPAMA has a template-based configurable structure. As any template structure, CPAMA has both fixed and configurable parts in its design. We have primarily considered supporting as many image processing algorithms as possible, while making a decision about whether a unit should be fixed or configurable. For instance, the number of ports of Constant Memory is fixed, however bit width of the address input of Constant Memory depends on the number of different constant instances. This is further explained in Fig. 7 and in Section 4.1.

Images are processed block by block on CPAMA. Each block size is equal to size of the processor array, i.e. *processor array height* × *processor array width*. Due to advantages of hardware & software co-design methods, CPAMA is designed in two parts; *hardware* and *software*. Conceptual architecture of CPAMA is shown in Fig. 1. The names in Fig. 1 are selected to present basic, abstract structure of CPAMA. In Fig. 1, upper dashed blocks are implemented on the same chip, that are hardware parts of a target design. However, the host processor in the lower dashed block can be implemented on or off the same chip. Software running on the host processor would be the software part of a target design. Global memory should be implemented on a separate chip in most cases due to its size. In this paper, modules presented as work items in Fig. 1 will be mostly referred as processors. Hence, the network that is composed of processor nodes will be called as processor array. Every work item has a private memory which may be registers that are available only for the work item itself. In addition, there is a local memory available for data sharing between the work item and global data cache. Local memory represents the FIFO registers of a processor.

To clarify which image/video processing algorithms are targeted, we would like to address classification of image processing algorithms that have been made earlier [27,28]. Although there are differences in expression of the classifications, these studies classify image processing algorithms into three categories:

1. *Point*: The output value at a specific coordinate is dependent only on the input value at that same coordinate.
2. *Local*: The output value at a specific coordinate is dependent on the input values in the neighborhood of that same coordinate.

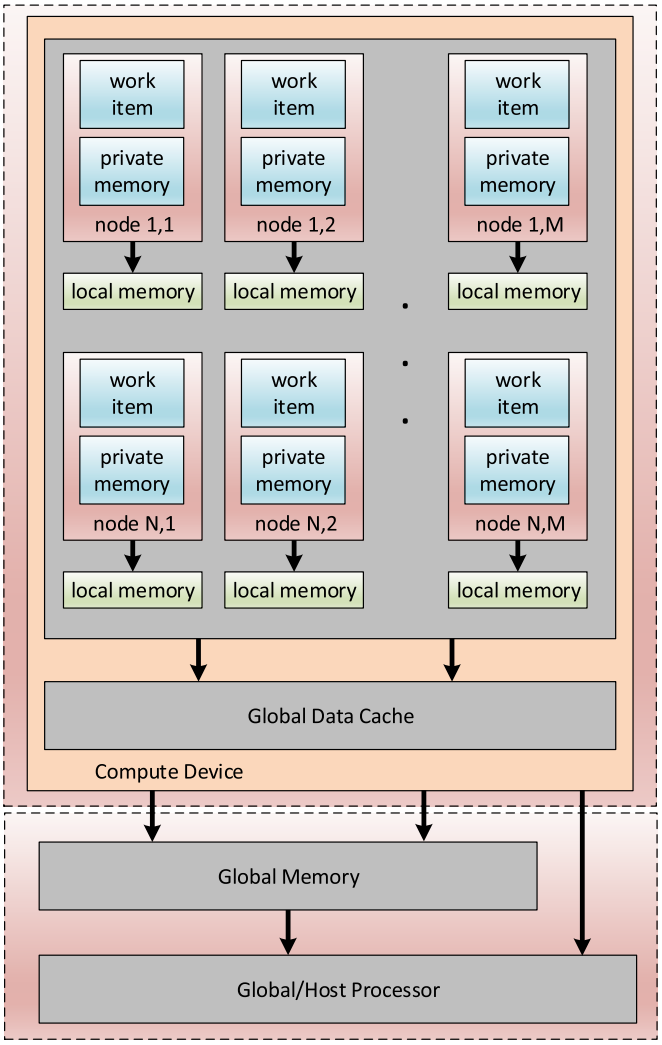

Fig. 1. Conceptual device architecture of CPAMA.

3. *Global*: The output value at a specific coordinate is dependent on all the values in the input image.

With CPAMA, we target to cover the algorithms described in 1 and 2 above. We do not focus on the third class in this study although it can be achieved through our architecture. Besides, one should note that; although the classification is given for still image processing algorithms, CPAMA supports video processing and image processing algorithms that are defined by more than one input image. This classification is presented only to demonstrate what kind of processing we are dealing with, regardless of the number of the input images or whether images/frames are received continuously.

In CPAMA, the whole raw image is assumed to be stored on a RAM to avoid standard image representations. A dedicated memory management unit is responsible for sending and receiving the blocks of an image. An image is assumed to be like the one in Fig. 2.

Surrounding pixels of a block are called neighboring pixels [29]. They may be needed in an algorithm which calculates the result pixel by its neighboring pixels, e.g. filtering algorithms. To explain basic parameters in our architecture, a sample algorithm is given in Eq. (1).

$$y_{i,j}[u] = \sum_{n=-r}^{r} c_1 * x1_{i+n,j+n}[u] + c_2 * x2_{i+n,j+n}[u] + \ldots \quad (1)$$

Throughout the text unless otherwise stated; r represents the neighborhood depth, x represents images (frames), c_i represents constants and u represents time. The number of different xs (e.g. $x1$, $x2$, etc.)

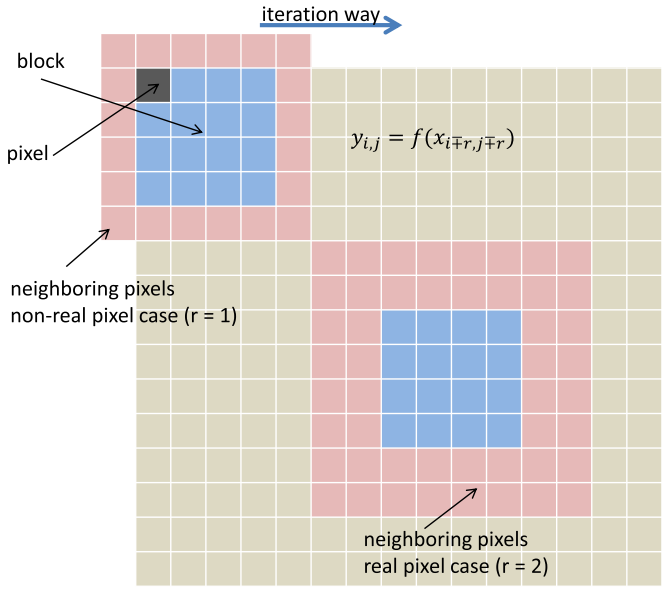

Fig. 2. Assumed image and primitive definitions of image processing.

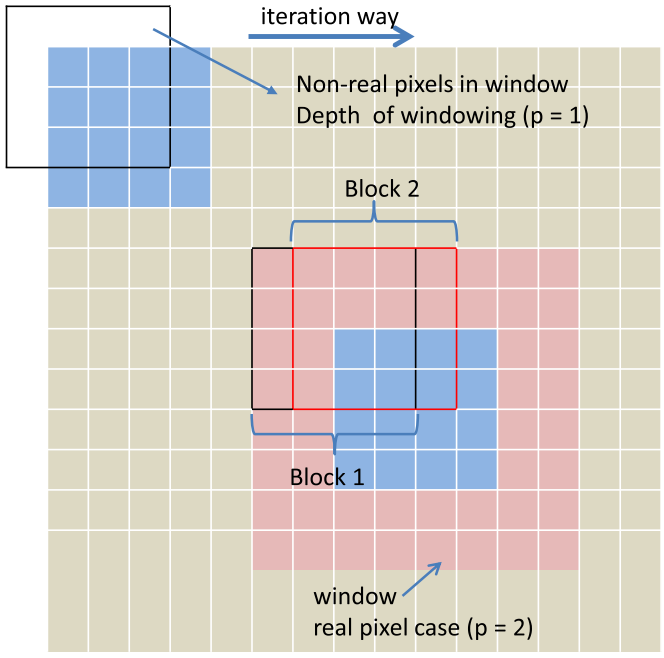

Fig. 3. Image processing using windows. p represents the window width.

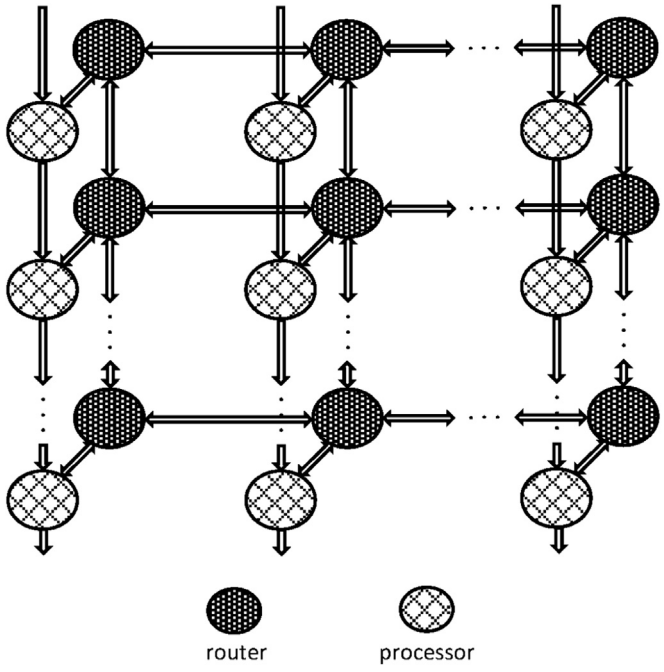

Fig. 4. Network on chip communication and basic blocks of hardware.

determines the number of images (frames) that are used in the algorithm. In filtering applications the number of images can be just one. However, in motion detection [30], and block-match [31] algorithms, there can be two or more successive frames of a video.

Depth of the neighborhood is indeed an important parameter, since it affects the amount of data to be sent. This parameter is equal to one for the upper block and two for the lower block in Fig. 2. If a series of algorithms are implemented on the same design, r only can be zero or the same positive value in each algorithm, because r affects the size of the FIFO which is explained in Section 4.1.1. However one can change it before compilation.

When considering local image processing algorithms, each block may not have real neighboring pixels like the upper block shown in Fig. 2. Mentioned block is on the boundary of the frame so its neighboring imaginary pixels should be chosen in a specific way. They may be chosen as their own values (zero-flux method), fixed value (e.g. zero), or pixels of a different block even in a different frame. The last option may be needed if the algorithm is performed by using more than one frame.

Imaginary pixels - non-real pixels - may be needed in other algorithms which use windows as well. Fig. 3 shows how an algorithm may need non-existent pixels when they do not use neighboring pixels. Window is basically a sub-block of an image which the algorithm is defined in. In Fig. 3, p represents the window width. Main difference between the algorithms that are defined by using neighboring pixels and windowing is the amount of required data. In an algorithm that is defined by an r neighborhood, processing one block of an image requires $(N + 2r)*(M + 2r)$ number of pixel data, where N and M are width and height of a block, respectively. However, when a window is used, $N*M$ number of pixel is enough to do the processing. The top left block in Fig. 3 shows the necessity of imaginary pixels when we do not use neighboring pixels.

As explained in Figs. 2 and 3, processing a block requires pixels of the block itself and some extra pixels due to neighboring, etc. To allocate the pixels required for the computation, a data structure is created. We can assume each image object has an accompanying data structure instance in the cache that stores the block that is ready to be processed. In order to support different neighboring configurations, we propose a FIFO communication that sends cache content to processor array. These hardware blocks are discussed in Section 4.1.1.

4. Hardware design

Hardware side of CPAMA consists of a 2D grid network structure as shown in Fig. 4. Considering the nature of image processing, there is a strong similarity between a 2D signal (image) and a 2D Mesh NoC. Therefore, we preferred this type of network in CPAMA.

One processor is connected to each node. Data communication among processors is done by routers. Image is delivered by FIFOs or routers through the network. FIFOs are placed in processors, and deliver the data in one (vertical) direction. Synchronization of the FIFOs is handled by global commands which are sent from host

P11 R0	P11 R1	P11 R2	P12 R0	P13 R0	P14 R0	P14 R1	P14 R2
P11 R3	P11 R4	P11 R5	P12 R1	P13 R1	P14 R3	P14 R4	P14 R5
P11 R6	P11 R7	P11 R8	P12 R2	P13 R2	P14 R6	P14 R7	P14 R8
P21 R0	P21 R1	P21 R2	P22 R0	P23 R0	P24 R0	P24 R1	P24 R2
P31 R0	P31 R1	P31 R2	P32 R0	P33 R0	P34 R0	P34 R1	P34 R2
P41 R0	P41 R1	P41 R2	P42 R0	P43 R0	P44 R0	P44 R1	P44 R2
P41 R3	P41 R4	P41 R5	P42 R1	P43 R1	P44 R3	P44 R4	P44 R5
P41 R6	P41 R7	P41 R8	P42 R2	P43 R2	P44 R6	P44 R7	P44 R8

Fig. 5. Pixel allocation method for a block of image. r = 2 on a 4 × 4 processor array.

processor. FIFO communication is built separately. In other words, delivering one block of an image may not be carried out by routers. In this way, processing elements on the network and FIFO elements, which are responsible from sending-receiving data, can operate concurrently.

We need to show how one block of an image is allocated among our processor array architecture. Fig. 5 explains the relation between the locations of pixels and the names of registers together with their hosting processors. Fig. 5 shows one block of an image. Each small square represents a pixel. Inner square (blue) represents center pixels of the processors. Outer (pink) pixels are neighboring pixels. $P_{i,j}$ represents the processors in the array. R_k represents the registers storing the related pixel. Each register stores one pixel. In short, each pixel is stored in a register R_k which is located in a processor $P_{i,j}$. From Fig. 5, one can see that each $P_{i,j}$ has to store different number of pixels, so each one of them has different number of registers. Processors which are not located on the boundaries have just one register to store one pixel of the related image.

For a better understanding, we show data communication through FIFOs in Fig. 6. The connections between FIFO registers (FR) are shown in Fig. 6. Here, coordinates (ids) of the processors are defined by colors and written at the bottom of each colored rectangle. So, registers with the same color belong to the same processor. For instance, red registers belong to the top left most processor, i.e. P11. The connections between the FIFO registers are in one vertical direction. Figs. 5 and 6 should be considered together. In both figures, $r = 2$, and we assumed our design to have a 4×4 processor array. We have mentioned that synchronization in the FIFO is handled by global commands. When the FIFO receives all the data of a block, global processor emits a command for copying content of the FIFO into registers of processors. Thus, processors get ready to operate, and FIFO gets ready to receive a new block. Further information about FIFO and the register file is given in Section 4.1.1.

We have a serial-to-parallel data converter at the input of the FIFO and a parallel to serial data converter at the output of the FIFO. In this way, we send and receive data serially to and from FIFO. Sending cache content to the FIFO needs to be done accordingly. In order to deliver the pixels to their corresponding locations, the pixel that should be sent first, has to be the last pixel of the block. We also provide multiple data input and output capability instead of a data converter. This option has to be chosen accordingly in CPAMA design. We left this feature optional to the user, because there might be no need to deliver all data in parallel in an application if computation takes more time than the communication. The area shaded with gray in Fig. 6 shows FIFO registers that are required for center pixels

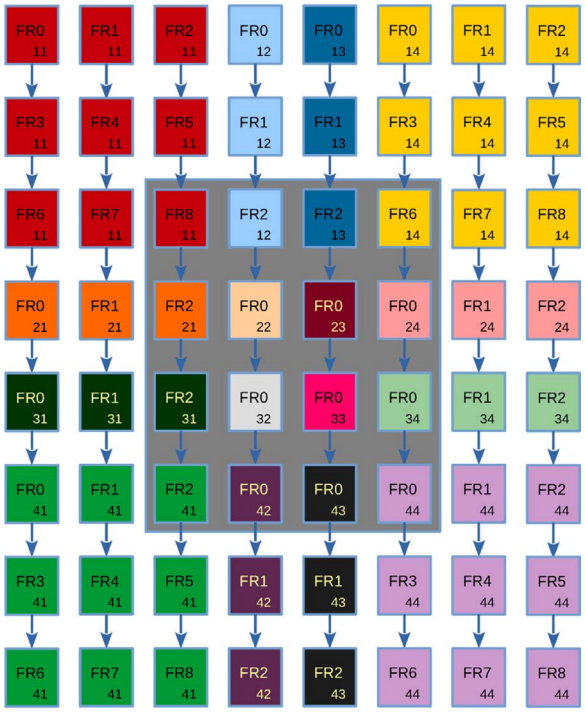

Fig. 6. Communication of FIFO registers. r = 2 on a 4 × 4 processor array.

of the processors. Rest of them are needed for neighboring pixels, when r is assumed as 2. Synchronization signals sent from software part are not shown here for the sake of simplicity.

4.1. Processor

The processor has been implemented as a *Very Long Instruction Word (VLIW)* architecture able to execute parallel instructions. It supports all the basic ADD, MULT, AND, JUMP, etc., instructions. Before giving our processor model, first we would like to discuss similarities between our processor and a conventional 32-bit processor (single cycle 32-bit MIPS) [32]. MIPS processor has a Program Counter (PC), Instruction Memory, Register File, ALU, Data Memory and MUXes for input selection related to these blocks. Our simplified processor model is presented in Fig. 7.

MIPS and the processor model proposed in this work have the same blocks except two differences: our model has a Constant Memory for storing constants and does not have a Data Memory.

MIPS Register File has two read data outputs and accompanying two selection inputs; one write data input and one accompanying selection input. MIPS architecture can only execute one ALU or register operation in one cycle. We design our processor such that it can execute ALU and register operations simultaneously in one cycle.

As seen in Fig. 7, ACC is a special purpose register (accumulator). One of its purposes is to provide concurrent ALU and register operations. When an ALU operation is being executed, a value can be fetched concurrently from PortIn and stored in a register. In order to make concurrent ALU and register MOVE operations, the register file has to have another read port. This feature is provided and optional in the architecture, but is not shown in Fig. 7 for the sake of simplicity.

Colored signals in Fig. 7 represent control signals derived from instruction bits and other signals. Bit width of almost any signal is variable. They vary depending on the application that is implemented. Number of the registers that are needed by the implementation determines the value of R in Fig Fig 7. The variable Z is determined by how many different instructions are used in the application. Therefore, this feature provides a variable opcode width. ALUSrc chooses the inputs of ALU unit; in other words, it acts like a selection signal of a MUX.

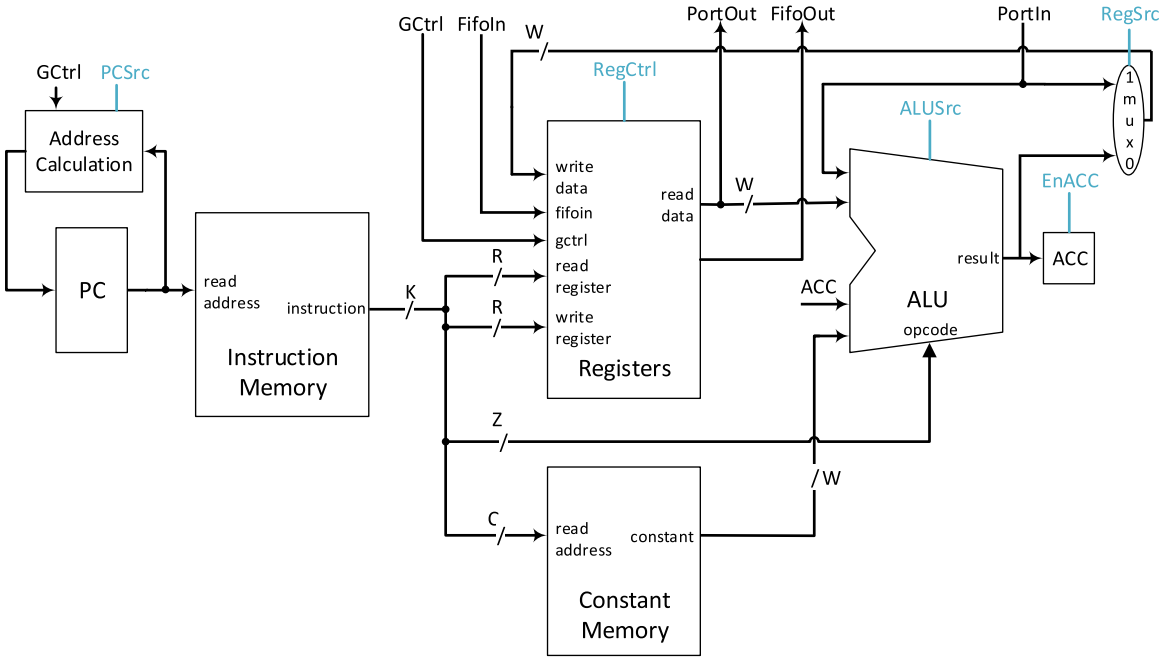

Fig. 7. Main blocks of the processor.

Constant values are not embedded in the instruction in our processor model. Instead, we provide each different constant an address and store them in a Constant Memory. For instance, if we had two 32-bit constants, we would address them with a single bit. Thus, 1 bit would occupy a space in the instruction word. Note that, this feature also provides the ability of using different precision for constants. Variable W represents the word length of the processor, i.e. precision of the ALU operations. Typically we are taking W as the width of constants. C is the address width of Constant Memory and varies depending on the number of different constant instances.

The address calculation of the program counter, PCSrc chooses either the very next address, the JUMP address, or the address that is delivered by FIFO. In the last case GCtrl command must be set accordingly for external address jump. This ability can be regarded as a function call, which is ordered by the host processor.

The processor has two kinds of communication. One of them is handled by FIFO and will be explained thoroughly in Section 4.1.1. The other communication, inter-processor communication, is handled through routers from the ports PortIn and PortOut. These ports have accompanying addresses which determines the destination of the packet. These addresses are not shown in Fig. 7 for simplicity.

All the blocks, wires and registers are instantiated on a need-to-have basis. Without redundant blocks, we may obtain an efficient circuit in terms of power consumption and area utilization.

Processors have some differences in their designs according to their locations. They are named as corner, edge and middle processors (middle processors are the ones that are not located on the boundaries of the array).

4.1.1. Register File and FIFO

Register File is designed to support register and ALU operations in one cycle. As shown in Fig. 8, the number of registers depends entirely on the application program. This prevents usage of redundant and unnecessary hardware. In Fig. 8 there are $\boldsymbol{n-k}$ registers for computation and $\boldsymbol{k}$ registers for FIFO. Besides, width and depth of the FIFO can change according to the place of the processor in the network. SI, EI, SO, EO are selection and enable signals for input and output of register file, respectively.

Recall that r is the neighborhood depth of the image processing algorithm and a is the argument number, i.e. the number of the frames

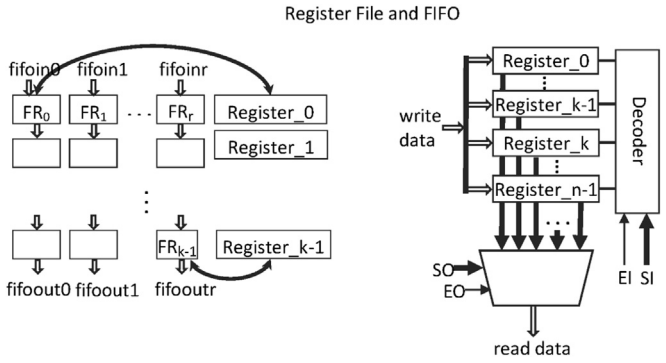

Fig. 8. Detailed model of Register File and FIFO. Data exchange between FIFO (FRs) and registers is synchronized by global command (GCtrl).

that are used to calculate one result frame;

- FIFO of the processors (FR) that is placed on the corners of the network has $(r+1)^2$ registers. Depth and width of the FIFO are $(r+1)$. For instance, top left registers in Fig. 6; these are the FIFO registers for P11 (processor 11).
- FR that is placed on the north and south borders of the network has $(r+1)$ registers. Width of the FIFO is 1 and depth of the FIFO is $(r+1)$. For instance, the registers colored with black at the bottom of Fig. 6; these are the FIFO registers for P43.
- FR that is placed on the west and east borders of the network has $(r+1)$ registers. Width of the FIFO is $r+1$ and depth of the FIFO is 1. For instance, the registers colored with orange at the left side of Fig. 6; these are the FIFO registers for P21.
- FR of the other processors has 1 register. Depth and width of the FIFO are 1. For instance, the brown register in the middle of Fig. 6; this is the FIFO register for P23.

Number of FIFO registers (FR) depends on the placement of the processors in the network, however number of the registers in register file that are related to the FIFO, i.e. k, depends on the argument number, i.e. a, as well. Note that a is taken as 1 in Fig. 8.

While designing the FIFO-register file structure, we have also considered power consumption in multi-port register files. As sug-

gested in studies [33,34], there will be a significant increase in power consumption of register files if the number of the ports of the register file increases. Especially when the size of the register file gets bigger, power consumption of the register file should be taken into account more seriously. It is suggested that power consumption will be proportional to N^3 where N is the number of the ports of the register file [34]. To conclude, register files of the processors should have as minimum number of ports and minimum size as possible. In our approach, sizes of register files are decided completely on a need-to-have basis. Middle processors can get their neigboring pixels from neighboring nodes through routers. Therefore, they just store their center pixels. However, processors on the edges have to store neighboring pixels since no other unit stores it. Keeping in mind that most of the nodes are placed in the middle, we think that this effective usage of registers yields us better results in terms of chip area and power consumption. Essentially, register files have two input (one for FIFO one for computation) and one output port. In case a type of processor needs an extra port for register file due to user program, an extra port will be generated only for that specific type of processor. Main goal here is to keep the register file capacity and the number of the ports small.

4.1.2. Arithmetic logic unit

In this architecture, arithmetic unit is designed as a template structure. It is designed to instantiate only the necessary operations. In future versions of CPAMA, we are planning to enhance its configurability by generating it from the algorithm definition.

In our arithmetic unit design, we have followed resource sharing approach similarly done in *multi-mode digital signal processing* [35,36]. For example, an ALU that can execute one addition and multiplication in one cycle, and as separate instructions, is designed sharing common hardware blocks.

In each target image processing application the implementation code may be different. Thus, the selected ALU operations may change. Therefore, we have designed a flexible instruction set that has variable opcodes.

4.2. Router

Routers shown in Fig. 4 have North, South, East, West and processor connections. A router basically decides which packet will be sent to which channel according to the address values accompanied with the packet. A packet consists of a pixel, a destination address and an argument number. In the network, it is assumed that there are no conflicts in communication, i.e., more than one packet is never sent to same port of a router at the same time. To eliminate possible temporary conflicts and make a stable system, a priority is assigned to each port of the router. Even if two channels try to send data to a processor at the same time, router will only pass the packet which comes from the channel that has higher priority. A schematic of the router is given in Fig. 9.

Router is basically composed of one multiplexer and one de-multiplexer unit. Data and address pair to be sent to next node is sent by the processor through the de-multiplexer unit. Incoming channels are selected by the multiplexer according to their priority and are delivered to the processor.

4.3. Reconfiguration in CPAMA

If the user wants to change between two or more programs at run-time due to chip area restrictions; at first, he/she has to have CPAMA's Assembler instantiated all the instructions and registers that are used in those programs in the processor architecture. Then, changing content of the Instruction and Constant memories will result in changing the program memory. If CPAMA is implemented on an FPGA, run-time programmability can be performed by partially reconfiguring memory blocks.

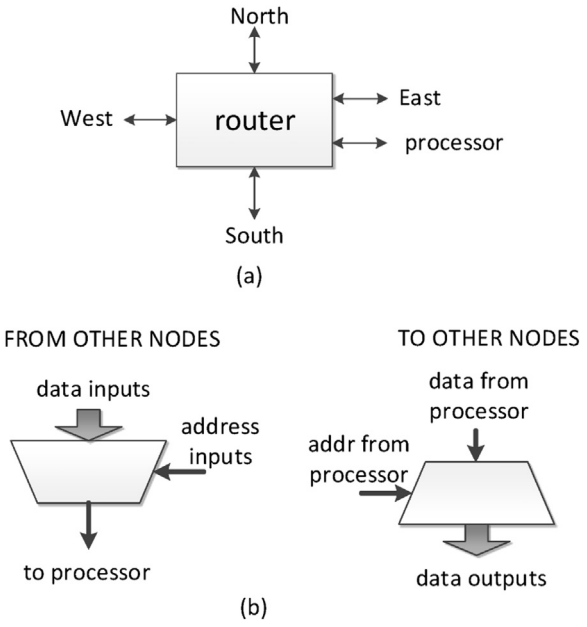

Fig. 9. (a) Inputs and outputs of the router are shown. Each channel (North, South, etc) has data and address input-output. (b) The relations between inputs and outputs of the router are shown.

As explained in [37,38], maximum bandwidth for configuration ports of a Xilinx Virtex-5 FPGA is 3.2 Gbps. Partial reconfiguration BIT file of a ROM consisting 64 × 32-bit words took 63 frames (smallest unit that can be reconfigured) in our experiment. Size of a frame in Virtex-5 is 41 32-bit words [39]. Note that, in our processors total Instruction and Constant memory size is normally smaller than this size (64 × 32-bit). Including fix parts of the BIT file, reconfiguration bit-stream length is 12,073 Bytes. Therefore, reconfiguration time of the memory block takes $(12,073 \times 8)bits \div 3.2Gbps = 30.2$ microseconds. The reconfiguration time scales fairly linearly as the partial BIT file size grows with the number of frames, with small variances depending on the location and contents of the frames [38]. Hence, total reconfiguration time of CPAMA changes linearly with respect to the number of the processors that are used in CPAMA.

In the ASIC case, the memory contents can be delivered through FIFO and written into the Instruction and Constant Memory. This method can be used in the FPGA as well. The architectural features related to programmability in ASIC case have not been implemented yet.

5. Case studies

We have evaluated performance of CPAMA by implementing four different algorithms; which are dot product, TIFF to gray level image transformation (TIFF2BW) [40], Inverse Discrete Cosine Transform (IDCT) and block-match.

5.1. Dot product

Dot product is the core of many image processing algorithms, e.g. filtering based image processing. According to the classification made in Section 3, dot product algorithm fits in the second group (local). We have implemented dot product on a Xilinx Virtex-5 FPGA (xc5vtx240t) using ISE 14.7 [41]. We have analyzed performance of CPAMA changing the number of processors in the network, both horizontally and vertically. We have evaluated 86 different configurations. Figs. 10 and 11 show how size of the network affects the area occupation; and Figs. 12 and 13 show how size of the network affects the period of the circuit.

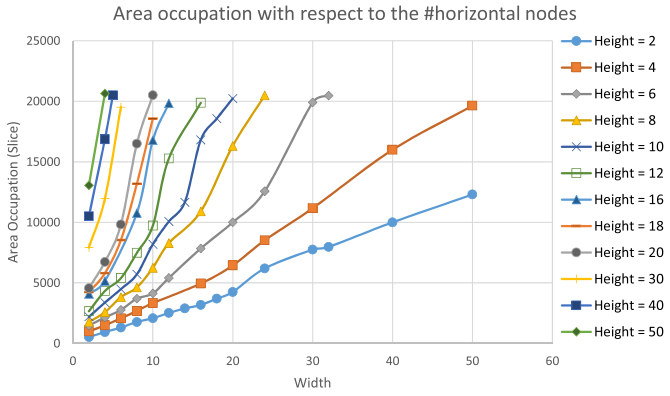

Fig. 10. Area occupation with respect to number of horizontal nodes.

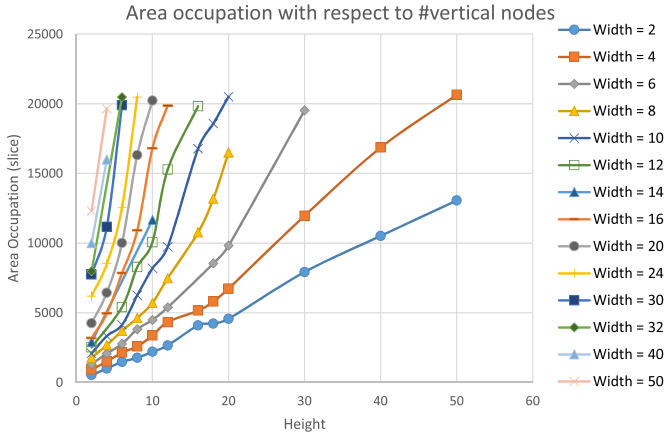

Fig. 11. Area occupation with respect to number of vertical nodes.

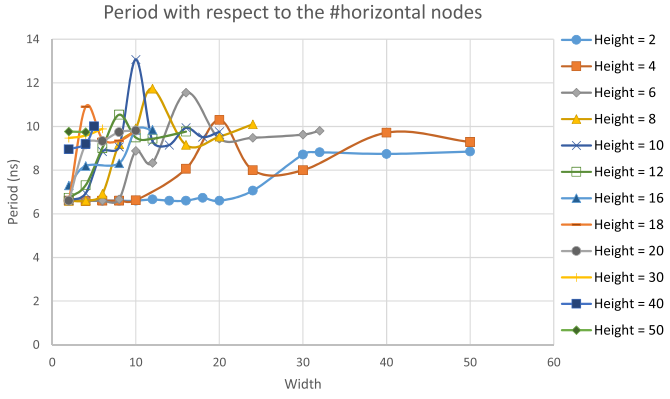

Fig. 12. Period with respect to number of horizontal nodes.

In these charts the neighborhood depth r is equal to 1. Height and width of the network are presented on the legend of the charts. Moreover, area and period results for a subset of the above configurations are given in Table 1 to express actual numbers. This time r = 1, 2. While doing this analysis, since manually doing each synthesis, place & route would take long, we have written a script to change network size and initiate the synthesize, place & route software. In the script, we adaptively change the period constraints to find the possible best value. However, the achievable best result for a given configuration might be better than the value that the script finds, because we had to limit the iteration count of the script due to long execution times. As seen in Table 1, our architecture is scalable. Table 1 also shows that area can be utilized more efficiently when the network is like a square, i.e *#horizontal nodes* ≈ *#vertical nodes*. Because, neighboring pixels are necessary to compute dot product. As explained in Section 4.1.1, a processor needs extra registers to store neigboring pixels

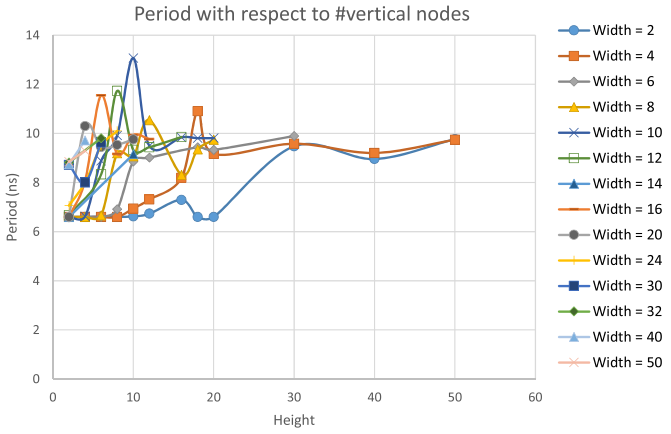

Fig. 13. Period with respect to number of vertical nodes.

Table 1
Performance results of several CPAMA configuration for dot product application.

#Processor	Width	Height	Period (ns)		Area (Slice)	
			r = 1	r = 2	r = 1	r = 2
16	2	8	6.599	6.626	1771	2466
	4	4	6.595	6.768	1496	2191
64	2	32	8.815	8.673	7976	8887
	4	16	8.059	9.212	4950	5965
	8	8	9.190	8.93	4628	5434
200	10	20	9.755	9.965	20,227	21,811
	4	50	9.290	9.418	19,635	22,646

when it is placed on the corner or edge of the network. When a network is close to square, it has fewer edge processors. Therefore, reducing the number of edge nodes yields a better result in terms of chip area occupation. Moreover, when the number of the processors gets large, the capability of synthesis, place & route tools are more dominant on the performance values. To calculate throughput of CPAMA, Eq. (2) can be used. This equation is valid for all kinds of applications that use FIFO for delivering the data, including dot product.

$$\begin{aligned} CT &= (NW + 2 \times r) \times (NH + 2 \times r)/BW \\ BT &= \max\{CT, PT\} + HS \\ fps &= 1/((IS/NS) \times BT \times T) \end{aligned} \tag{2}$$

Terms in Eq. (2) represent the following:

- IS: Image size, (width of the image) × (height of the image)
- NH: Network height, height of the processor array
- NW: Network width, width of the processor array
- NS: NW × NH
- r: neighborhood depth
- BW: Bandwidth between the processor array and data cache in terms of ”pixel per cycle”
- CT: Communication time, cycle count that is spent for delivering pixels to the network
- PT: Process time, cycle count that is spent by the processor array performing instructions
- HS: Handshaking delay in terms of cycles, typically 3
- T: period, duration of one cycle in seconds
- fps: frame per second
- BT: Cycle count that is needed to process a block. Time that is spent for handshaking is included.

Eq. (2) implies that CPAMA's throughput is determined by either computation or communication time. Since computation and communication are performed concurrently on the processor array, the longer latency determines throughput of the architecture. If process time is

Table 2
Comparison of performance values of CPAMA and ADRES for TIFF2BW application.

Architecture	Frequency (MHz)	Throughput (pixel/us)	Energy (mJ)	Area (mm^2)
ADRES 4 × 4	300	303	0.54	NA
CPAMA 4 × 4*	300	400	0.37	0.40
CPAMA 4 × 4	350	466	0.40	0.42
CPAMA 8 × 8	333	1641	0.54	1.45

less than communication time, which is usually the case especially for large networks, then CPAMA works like a stream processor. In other words, it produces the result pixel/data as soon as it gets a new pixel/data. Furthermore, if bandwidth between data cache and processor array is larger than 1 pixel per cycle, this will affect the throughput favorably.

5.2. TIFF2BW

We have implemented TIFF2BW application on CPAMA to make a comparison with the performance values of ADRES architecture [6]. We have chosen ADRES because it is not dependent on a specific device like RAMPSoC [19]. In addition, ADRES is well analyzed as a CGRA architecture in literature [2]. More importantly the TIFF2BW test that was done on ADRES [6] is repeatable. TIFF2BW application fits in the first group of image processing applications (point) mentioned in Section 3. We have implemented three different CPAMA instances by using the same CMOS technology (90 nm) as ADRES. To make a fair comparison between two architectures, we have selected the same configurations, e.g. frequency, array size, precision (32-bits), etc. Our results are shown in Table 2. Here, we refer the CPAMA instance (CPAMA 4 × 4*) which has the same configurations as ADRES. The other instances in Table 2 are presented to show the performance of CPAMA for different size of arrays and for different frequencies.

We have implemented the design by using TSMC90GP standart cell library [42] and Cadence [43] tools. First, we have synthesized VHDL code of CPAMA by Cadence's RTL Compiler. Synthesized circuit was placed and routed automatically by Cadence's Encounter. Placed and routed design worked at a frequency of 300 MHz. We have made a back annotated simulation at 300 MHz to obtain switching activities for CPAMA. Simulation was performed by Cadence's NCSim. When generating switching activity file, we have used the same picture obtained from [40] as the input for CPAMA, thus we eliminated the effect of input on power measurement. Measurement of dynamic power consumption of CPAMA was done by Cadence's Encounter. As a result, Encounter measured 65.35 mW for dynamic power consumption of CPAMA. This number includes power consumption of all the parts of the 4×4 processor array. Since global memory and host processor are not part of the processor array, they are not included in power measurement. Similar exclusions were made in the compared study [6] too. Dynamic power of ADRES is 71.69 mW for TIFF2BW application [6]. TIFF2BW application for an 1520 by 1496 image can be performed in 1.71 million cycles on 4×4 CPAMA however the best value 4 × 4 ADRES can achieve is 2.25 million cycles for the same size image. We can measure the energy that the architectures consume as follows:

$$Energy = cyclecount \times clockperiod \times power.$$

According to our comparison regarding TIFF2BW application, CPAMA architecture consumes 31% less energy, and provides 32% more throughput than ADRES.

5.3. Inverse DCT

We have implemented 2D inverse DCT (IDCT) algorithm too, to compare the performance of our architecture with other ADRES implementations [44]. In this ADRES architecture, pipeline mechanism is enabled and the register file structure is changed. Both 4 × 4 and 8 × 8 array architectures are implemented and their performance results are given. These ADRES instances are implemented by using a 90 nm low power CMOS library to lower the power consumption.

To make a fair comparison with this study [44], we implemented 4 × 4 and 8 × 8 CPAMA instances by using TSMC90LP (low power) library.

One should note that, IDCT algorithm does not fit into the two categories (point, local) that are mentioned in Section 3. As mentioned, although we focus on image/video algorithms that are in the first and second category, other applications may still be done. Here, by implementing IDCT, we give an example to the third category. While implementing IDCT, we used similar partitioning and matrix multiplication approaches to the studies in literature [45,46]. Details of the ASIC implementation are similar to the TIFF2BW implementation. So, they are not repeated here.

Since the method followed for IDCT implementations of ADRES is not stated, we have implemented IDCT on CPAMA by using two different methods. In this way, we aim to demonstrate two features of CPAMA: 1) Algorithm selection truly effects performance of CPAMA. 2) CPAMA is scalable. In 4 × 4 CPAMA instance, we implemented IDCT by using row-column decomposition, i.e. two 1D-IDCT. In 4 × 4 CPAMA, we selected Chen-Wang [45] approach for 1D-IDCT implementation. On the other hand, for 8 × 8 CPAMA instance, we used usual 8 × 8 matrix multiplication instead of Chen-Wang approach. We selected cross-wired mesh array [46] approach for matrix multiplication. This approach is proposed to multiply two variable matrices. However in IDCT, one multiplier is always constant. Hence, by re-arranging the array structure of the cross-wired mesh array, we mapped constant-variable matrix multiplication onto CPAMA without cross connections. In 8×8 CPAMA instance, we delivered the input data through routers by doing a minor modification. We could have used FIFO as usual for delivering the data; but, in this instance, control of the network is easier this way.

Performance values of CPAMA and ADRES are compared in Table 3.

In Table 3, throughput is given in terms of block/us. This is the number of 8 × 8 blocks that are calculated in one micro second. In ADRES implementations, the IDCT execution time is given for 396 units of 8 × 8 blocks. So, in Table 3, the throughput column is calculated dividing 396 by execution time values.

Execution time and energy values for 8 × 8 ADRES implementation are taken from the graph shown in [44]. Because their exact values are not given in that study.

As shown in Table 3, for IDCT application, 8 × 8 CPAMA is more efficient than 8 × 8 ADRES in terms of energy consumption and throughput. CPAMA provides 2.4× more throughput and consumes 50% less energy than ADRES in this configuration. On the other hand, for 4 × 4 array size, ADRES is more efficient than 4 × 4 CPAMA in terms of energy consumption and throughput, except area occupation. This time, CPAMA provides 56% less throughput and consumes 55% more energy than ADRES. According to Table 3, it can be deduced that scalability is not an issue for CPAMA. Otherwise, the throughput

Table 3
Comparison of performance values of CPAMA and ADRES for IDCT application.

Architecture	Frequency (MHz)	Throughput (block/us)	Energy (uJ)	Area (mm^2)	Technology
CPAMA 4 × 4	400	0.74	29.6	0.39	90 nm LP
ADRES 4 × 4	312	1.70	19.1	1.08	90 nm LP
CPAMA 8 × 8	303	5.60	21.7	1.41	90 nm LP
ADRES 8 × 8	294	1.65	43.2	NA	90 nm LP

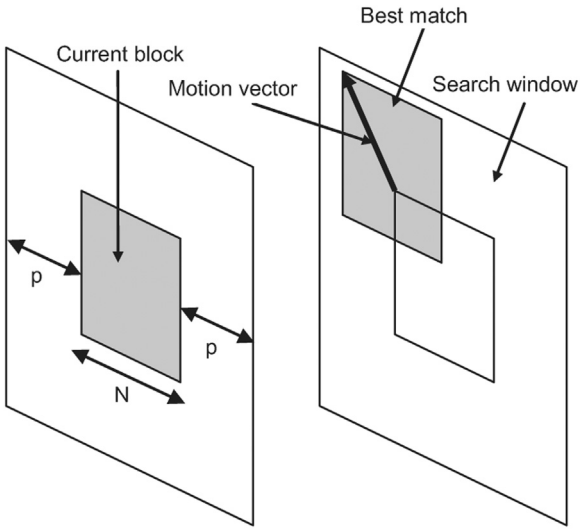

Fig. 14. Block-matching and computation of motion vector.

Table 4
CPAMA instances for block-match application.

Architecture	Frequency (MHz)	Throughput (block/us)	Area (slice)	#Pixels in a block
CPAMA 4 × 4	140	6.94	1797	16
CPAMA 8 × 8	125	3.45	7233	64
CPAMA 16 × 16	83	1.21	26,575	256

wouldn't be higher for the larger array. The method used in implementing the application has direct effect on the performance of the architecture. Second method that we have used is more suitable to implement on CPAMA.

There is a significant difference in throughput between two CPAMA instances. Because, in the first approach that is used for 4 × 4 CPAMA, 1D-IDCT is applied to each row and then column one by one. This spends a significant amount of handshaking time. Besides, the processor array cannot be utilized well while using this method. On the other hand, the second method that is used for 8 × 8 CPAMA needs some hand work to be efficiently mapped.

5.4. Block-match application

We have implemented block-match algorithm as a proof-of-concept for multiple frames. Block-match algorithm is used in video encoding. Since it requires extensive computation, it is generally implemented as a dedicated hardware unit [31,47]. Fig. 14 presents how block-match algorithm works and motion vectors are computed. In Fig. 14, N represents the block size and p represents the search window size. In our application, the sum of absolute differences (SAD) of pixels is computed to find the similarity between two blocks. Three different instances of CPAMA are presented in Table 4. Size of the blocks for matching algorithm is taken the same as the network size. The word-length of the processors are selected as 16-bits in each configuration. Frames are delivered to the network as multiple words, i.e., one row of data is fed to the network at a time. Each instance is implemented on a Xilinx Virtex-5 FPGA (xc5vtx240t) using ISE 14.7. Changing configuration of these three CPAMA instances is managed by only changing the size of the network parameter.

In Table 4, CPAMA 4 × 4 has a better throughput, but image block size processed by CPAMA 4 × 4 is smaller than that of CPAMA 8 × 8 and CPAMA 16 × 16 configurations.

6. Conclusion

Our proposed architecture CPAMA is a highly configurable processor array targeted for low power, low cost image/video processing devices. In comparison with ADRES, CPAMA has shown better performance in TIFF2BW and comparable performance in IDCT application in terms of energy consumption, throughput and area occupation. We think, this is because it occupies only the necessary hardware for a given application. This is achieved by considering the image processing nature in every development cycle of CPAMA.

In the first and second group of multimedia processing applications (point and local), CPAMA is quite reusable and easily configurable. For these application groups, configuration can be done by just changing the parameters of the array or/and processor program. In consumer electronics, improving time-to-market of a low cost and low power image/video processing chip is a significant goal. Due to re-usability of our design, design and verification cycle of an implementation using CPAMA will be shorter. In addition, we have a toolchain project in its final stages to automatically generate design files of the CPAMA. This toolchain also accelerates the design process. Consequently, we think CPAMA is a good candidate for consumer devices that exploit image/video processing tasks.

Acknowledgment

The authors would like to thank Mr. Gökhan Işık for his recommendations on ASIC implementation, and Dr. Salih Bayar for his help on partial reconfiguration techniques in FPGAs.

References

[1] D. Macmillen, R. Camposano, D. Hill, T. Williams, An industrial view of electronic design automation, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 19 (12) (2000) 1428–1448. http://dx.doi.org/10.1109/43.898825.

[2] B. De Sutter, P. Raghavan, A. Lambrechts, Coarse-grained reconfigurable array architectures, in: S.S. Bhattacharyya, E.F. Deprettere, R. Leupers, J. Takala (Eds.), Handbook of Signal Processing Systems, Springer, US, 2010, pp. 449–484. http://dx.doi.org/10.1007/978-1-4419-6345-1_17.

[3] 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: P. Y. K. Cheung, G. Constantinides (Eds.), Field Programmable Logic and Application, Vol. 2778 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, pp. 61–70. ⟨http://dx.doi.org/10.1007/978-3-540-45234-8_7⟩.

[4] D. Gohringer, M. Hubner, J. Beck, Adaptive multiprocessor865 system-on-chip architecture: new degrees of freedom in systemdesign and runtime support, in: M. Hubner, J. Becker (Eds.), , Multiprocessor System-on-Chip, Springer, New York, 2011, pp. 127–151. http://dx.doi.org/10.1007/978-1-4419-6460-1_6.

[5] S. Pedre, T. Krajnk, E. Todorovich, P. Borensztejn, Accelerating embedded image processing for real time: a case study, J. Real-Time Image Process. (2013) 1–26. http://dx.doi.org/10.1007/s11554-013-0353-2.

[6] M. Hartmann, V. Pantazis, T. Vander Aa, M. Berekovic, C. Hochberger, Still image processing on coarse-grained reconfigurable array architectures, J. Signal Process. Syst. 60 (2) (2010) 225–237. http://dx.doi.org/10.1007/s11265-008-0309-0.

[7] B. Stabernack, K.-I. Wels, H. Hubert, A system on a chip architecture of an h.264/avc coprocessor for dvb-h and dmb applications, IEEE Trans. Consum. Electron. 53 (4) (2007) 1529–1536. http://dx.doi.org/10.1109/TCE.2007.4429248.

[8] B. Mei, M. Berekovic, J.-Y. Mignolet, Adres & dresc: Architecture and compiler for coarse-grain recon gurable processors, in: S. Vassiliadis, D. Soudris (Eds.), Fine- and Coarse-Grain Recon- gurable Computing, Springer, The Netherlands, 2007, pp. 255–297. http://dx.doi.org/10.1007/978-1-4020-6505-7_6.

[9] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, B. Hutchings, A reconfigurable arithmetic array for multimedia applications, in: Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, FPGA '99, ACM, New York, NY, USA, 1999, pp. 135–143. ⟨http://dx.doi.org/10.1145/296399.296444⟩.

[10] C. Jang, J. Kim, J. Lee, H.-S. Kim, D.-H. Yoo, S. Kim, H.-S. Kim, S. Ryu, An instruction-scheduling-aware data partitioning technique for coarse-grained reconfigurable architectures, in: Proceedings of the 2011 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, LCTES '11, ACM, New York, NY, USA, 2011, pp. 151–160. ⟨http://dx.doi.org/10.1145/1967677.1967699⟩.

[11] N.R. Miniskar, R.R. Patil, R.N. Gadde, Y.C.R. Cho, S. Kim, S.H. Lee, Intra mode power saving methodology for cgra-based reconfigurable processor architectures, in: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 714–717. ⟨http://dx.doi.org/10.1109/ISCAS.2016.7527340⟩.

[12] H. Eichel, Customising a processor architecture for multimedia applications,

Electron. Syst. Softw. 1 (4) (2003) 29–33. http://dx.doi.org/10.1049/ess:20030406.

[13] J.-C. Chu, C.-W. Huang, H.-C. Chen, K.-P. Lu, M.-S. Lee, J.-I. Guo, T.-F. Chen, Design of customized functional units for the vliw-based multi-threading processor core targeted at multimedia applications, in: 2006 IEEE International Symposium on Circuits and Systems, 2006, pp. 2389–2392. ⟨http://dx.doi.org/10.1109/ISCAS.2006.1693103⟩.

[14] C.S. Bassoy, H. Manteuffel, F. Mayer-Lindenberg, Sharf: An fpga-based customizable processor architecture, in: 2009 International Conference on Field Programmable Logic and Applications, 2009, pp. 516–520. ⟨http://dx.doi.org/10.1109/FPL.2009.5272447⟩.

[15] K. Masselos, F. Catthoor, C. E. Goutis, H. DeMan, Low power mapping of video processing applications on vliw multimedia processors, in: IEEE Alessandro Volta Memorial Int. Workshop on Low Power Design, 1999, pp. 52–60.

[16] K. Sanghai, R. Gentile, Multi-core programming frameworks for embedded multimedia applications, 2017. https://www.ll.mit.edu/HPEC/agendas/proc07/Day3/17_Sanghai_Abstract.pdf.

[17] M. Rashid, L. Apvrille, R. Pacalet, Application specific processors for multimedia applications, in: 2008 11th IEEE International Conference on Computational Science and Engineering, 2008, pp. 109–116. ⟨http://dx.doi.org/10.1109/CSE.2008.26⟩.

[18] Synopsys, 2017. ⟨http://www.synopsys.com⟩.

[19] D. Göhringer, J. Becker, High performance reconfigurable multi-processor-based computing on fpgas, in: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, 2010, pp. 1–4. ⟨http://dx.doi.org/10.1109/IPDPSW.2010.5470800⟩.

[20] M. Tukel, M. Yalcin, A new architecture for cellular neural network on reconfigurable hardware with an advance memory allocation method, in: Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on, 2010, pp. 1–6. ⟨http://dx.doi.org/10.1109/CNNA.2010.5430316⟩.

[21] N. Yildiz, E. Cesur, K. Kayaer, V. Tavsanoglu, M. Alpay, Architecture of a fully pipelined real-time cellular neural network emulator, IEEE Trans. Circuits Syst. I: Reg. Pap. 62 (1) (2015) 130–138.

[22] S. Malki, L. Spaanenburg, A cnn-specific integrated processor, EURASIP J. Adv Signal Process. 2009 (2009) 1–14.

[23] Z. Voroshazi, Z. Nagy, A. Kiss, P. Szolgay, Implementation of embedded emulated-digital cnn-um global analogic programming unit on fpga and its application, International J. Circuit Theory Appl. 36 (2008) 589–603.

[24] Stp engine ip core, 2017. ⟨https://www.renesas.com/en-us/products/programmable/stp-engine.html⟩.

[25] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Toi, T. Awashima, Implementation and evaluation of aes/adpcm on stp and fpga with high-level synthesis, in: SASIMI 2015 Proceedings, 2015, pp. 415–420.

[26] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Toi, T. Awashima, Stream applications on the dynamically reconfigurable processor, in: Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921), 2004, pp. 137–144. ⟨http://dx.doi.org/10.1109/FPT.2004.1393261⟩.

[27] Fundamentals of image processing, 2017. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUDELFT/FIP2_3.pdf.

[28] G.A. Baxes, Digital Image Processing: principles and Applications, 1st edition, Wiley, USA, 1994.

[29] R.C. Gonzalez, R.E. Woods, Digital Image Processing, 2nd edition, Prentice Hall, New Jersey, USA, 2002.

[30] S.-C. Huang, An advanced motion detection algorithm with video quality analysis for video surveillance systems, IEEE Trans. Circuits Syst. Video Technol. 21 (1) (2011) 1–14. http://dx.doi.org/10.1109/TCSVT.2010.2087812.

[31] K.K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation, 1st edition, John Wiley & Sons, USA, 1999.

[32] D.A. Patterson, J.L. Hennessy, Computer Organization and Design, 4th edition, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2008.

[33] V. Zyuban, P. Kogge, The energy complexity of register files, in: Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on, 1998, pp. 305–310.

[34] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, J. Owens, Register organization for media processing, in: High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on, 2000, pp. 375–386. ⟨http://dx.doi.org/10.1109/HPCA.2000.824366⟩.

[35] V.V. Kumar, J. Lach, Highly flexible multimode digital signal processing systems using adaptable components and controllers, EURASIP J. Appl. Signal Process. 2006 (2006) 1–9.

[36] C. Chavet, C. Andriamisaina, P. Coussy, E. Casseau, E. Juin, P. Urard, E. Martin, A design flow dedicated to multi-mode architectures for dsp applications, in: Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on, 2007, pp. 604–611. ⟨http://dx.doi.org/10.1109/ICCAD.2007.4397331⟩.

[37] S. Bayar, A. Yurdakul, A dynamically reconfigurable communication architecture for multicore embedded systems, J. Syst. Archit. 58 (3–4) (2012) 140–159. http://dx.doi.org/10.1016/j.sysarc.2012.02.003.

[38] Xilinx, Partial Reconfiguration User Guide.

[39] Xilinx, Virtex-5 FPGA Configuration User Guide.

[40] Mibench: Embedded benchmark suite, 2017. ⟨http://wwweb.eecs.umich.edu/mibench⟩.

[41] Xilinx, 2017. http://www.xilinx.com/products/design-tools/ise-design-suite.html.

[42] Taiwan semiconductor manufacturing company, 2017. http://www.tsmc.com.

[43] Cadence, 2017. ⟨http://www.cadence.com⟩.

[44] F. Bouwens, M. Berekovic, B. De Sutter, G. Gaydadjiev, Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 66–81. ⟨http://dx.doi.org/10.1007/978-3-540-77560-7_6⟩.

[45] ChenWang, Inverse two dimensional dct, in: Proceedings of the IEEE ASSP-32, 1984, pp. 803–816.

[46] S. Kak, Efficiency of matrix multiplication on the cross-wired mesh array, 2017. arXiv:1411.3273.

[47] S. Bayar, A. Yurdakul, M. Tukel, A self-reconfigurable platform for general purpose image processing systems on low-cost spartan-6 fpgas, in: 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), 2011, pp. 1–9. ⟨http://dx.doi.org/10.1109/ReCoSoC.2011.5981513⟩.