A Novel Template-based Multimedia Processor Array and Its Toolset

Mehmet Tükel[1], Arda Yurdakul[2], Berna Örs[1]

[1] Department of Electronics and Communication Engineering, Istanbul Technical University
tukel@itu.edu.tr, orssi@itu.edu.tr
[2] Department of Computer Engineering, Boğaziçi University
yurdakul@boun.edu.tr

Abstract

In this paper, we propose a template-based multimedia processor array and its design framework. The configurable processor array is designed for low-cost, low-power image/video processing applications. Each processor in the array is template-based and implemented by considering the nature and specifications of image/video processing domain. The framework can set the size of the network and the parameters of the building blocks of each processor. Hence, the generated architecture occupies only the necessary amount of logic. To show the scalability and the performance of the design, different instances of the architecture implementing four test applications are generated. We have obtained better or comparable results in terms of energy consumption, throughput and area occupation compared to that of the similar architectures in literature.

1. Introduction

In today's technology, we encounter various processor chips that provide multimedia functionalities for different application domains such as, hand-held devices, smart watches, industrial applications. Since the specifications of each domain are different, it is hard to propose a generic processor for every domain. Architectures that can be customized for different needs are proposed as a solution to this problem [1]. To meet the application requirements on these architectures, number and behaviour of the processing elements, and size of the memory units are left configurable. In this paper, we propose a Customizable Embedded Processor Array for Multimedia Applications (CPAMA). The contribution of the proposed architecture is that in each development cycle of CPAMA, the nature of image/video processing domain is considered.

To make the differences clear between the proposed architecture and the existing architectures, we categorized the studies in literature into three groups. First group is Coarse Grain Reconfigurable Arrays (CGRA), with which the proposed architecture has a similarity in design approach. Second group covers the architectures that are tailored for a specific image/video application. Third group comprises the architectures that are dependent on or targeting a specific device.

The difference between the proposed architecture and the CGRAs [1, 2, 3, 4] can be pointed out in three items. First, they are composed of configurable ALUs. Second, they pass data among ALUs through multi-port register files. Instead, CPAMA is composed of fully configurable processors. Moreover, data communication is handled through routers and FIFO instead of register files. Third, the problem defined in [2] is a loop expansion problem. However, our proposed architecture is defined using the concepts in image/video processing domain, e.g. neighborhood, number of frames. The details about these concepts are covered in Sec. 2.

A considerable part of the studies we reviewed are in the second group. These studies could be regarded as configurable application specific processors. In these architectures, the target application may be filtering [5, 6], video encoding [7, 8], or a specific image processing algorithm [9]. As opposed to the mentioned application specific processors, in CPAMA, the target application is not a specific one. One can easily map an image/video processing algorithm onto the proposed architecture if the result pixel at a specific coordinate is computed by the input pixel of the same coordinate or by the pixels in the neighborhood of that same coordinate.

The third category includes the studies that are dependent on or targeting a specific type of device. Runtime Adaptive Multi-Processor System-on-a-Chip (RAMPSoC) architecture [10] exploits partial reconfigurability feature of the FPGAs. This feature is not supported by all FPGA vendors. Therefore, RAMPSoC can be regarded as a multi-processor system that can be ported to partially reconfigurable FPGAs. Another architecture called SHARF [11] targets FPGAs as well, but it is not necessarily dependent on them. However, the processors of the architecture are tightly coupled to the controller unit, which may be a bottleneck when the number of the processors gets larger. The proposed architecture CPAMA is not dependent on a specific device or a design software. Since it is purely written in VHDL, it can be implemented on an FPGA or as an ASIC.

The paper is organized as follows: In Sec. 2, basic concepts that we used in designing CPAMA are covered. Sec. 3 details the design of the proposed architecture. In Sec. 4, a tool-chain for CPAMA is presented. Results of four different test applications are presented and compared to the existing similar architectures in literature in Sec. 5. Final section concludes the paper and we make our remarks about the study.

2. Basic Concepts in CPAMA

CPAMA is designed as a template-based configurable architecture. Therefore, CPAMA has both fixed and configurable parts. Some parameters are defined as configurable or fixed, is to support as many applications as possible.

In CPAMA, an image/frame is processed block by block. The size of the block is equal to the size of the processor array in CPAMA. Fig. 1 presents a sample image. Each small square represents a pixel. The blue squares represent the pixels in a block of an image. The pink squares surrounding the blue ones represent the neighboring pixels. In Fig. 1, r represents the neighborhood depth. Unless otherwise stated, r represents the neigborhood depth throughout the text. For the top left block, r is selected 1; and for the lower block, r is selected 2. A neighboring pixel can be a real pixel as well as a non-real pixel. When a pixel is on the boundaries of the image and $r > 0$, the result pixel has to be computed using the values that do not exist. A fixed value or the value of the nearest pixel can be selected in these circumstances, depending on the image processing algorithm. In CPAMA, we have designed the communication network and the processors to support neighboring pixels. In literature [12, 13], despite some minor

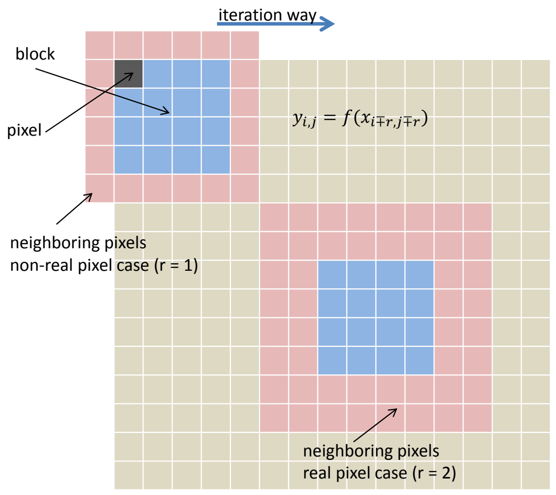

Figure 1. Assumed image and primitive definitions of image processing.

differences, image processing algorithms are categorized into three groups:

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

In this study, although we focus on the first two groups, the third group of applications can be achieved on our architecture as well. Note that, the above classification is given for still image processing. However on CPAMA, image and video processing algorithms with more than one input image can be implemented. We simply mention this classification to state what kind of algorithms we focus on, regardless the number of images, or whether the images/frames are received continuously.

3. Basic Blocks Of CPAMA

CPAMA is designed considering hardware and software co-design approach. In the proposed architecture, host processor is responsible for global commands, and the hardware part, i.e. processor array, is responsible for processing the data.

CPAMA is built on a grid network-on-a-chip (NoC). As shown in Fig. 2, each node consists of a processor and a router. Data communication in the network can be done in two ways: 1) data can be sent through FIFOs of the processors in vertical direction, 2) data can be sent from a processor to the nearest

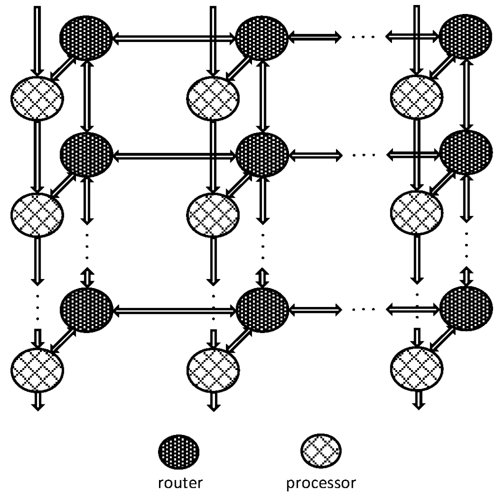

Figure 2. NoC communication and basic blocks of hardware.

processors through routers. Global commands are not drawn for the sake of simplicity.

3.1. Processor

Each processor in the array has a template-based structure. As opposed to a classical processor e.g., MIPS, the proposed processor has a memory for storing constants and does not have a data memory. As shown in Fig. 3; the size of each block, widths of wires and registers are all configurable and represented by a letter, e.g., W, R, Z. This way the sub-blocks can be designed on a need-to-have basis.

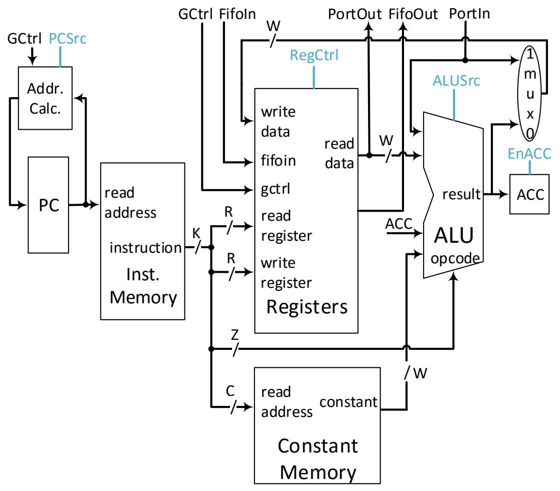

Figure 3. Main blocks of the processor.

In Fig. 3, signals colored with blue represent the internal control signals. PC represents the program counter, and it is calculated using GCtrl and PCSrc signals. GCtrl is delivered by the host processor and acts as a function call. PCSrc is a control signal. In Adress Calculation sub-block, the next value of the program counter is determined. Instruction Memory stores the instructions. According to the user program, each processor might have a different instruction memory. An instruction is composed of register addresses, opcode, constant memory address, etc. Therefore, the size of the instruction (K) depends on the size of the mentioned signals. Each different constant value in the user program is given an address and stored in the Constant Memory. This feature enables using different precisions for constants. Typically, we have selected the same precision (W) for constants and the registers. The register file is built supporting the neighboring values of a block. The FifoIn input receives the incoming pixels from the above processor, and the FifoOut signal sends the pixels to the below processor. PortIn and PortOut receives and sends data from and to the router, respectively. Depending on the location of the processor in the array and the neighborhood depth, size of the FIFO registers can vary. For instance; for $r = 1$, if the number of the frames is 1 and if the processor is located on the top left most node, the number of FIFO registers will be 4. For the same case, the FIFO registers of the processor located in the middle of the array will be 1. Number of the registers, which are used for temporary storage, is determined by the user program. Hence, the size of the register file is determined by the sum of FIFO registers and the registers that are used for temporary storage. The ALU is the arithmetic logic unit which executes the operations. The opcode width (Z) depends on the number of the different instructions that are used in the user program. Besides, the ALU instantiates only the used instructions. Therefore, it occupies only the necessary amount of logic. ACC is the accumulator which stores the output of the ALU.

3.2. Router

As shown in Fig. 4, a router consists of a multiplexer and a de-multiplexer. A router has North, South, East, West and processor channels. An incoming packet has an argument number, the pixel data and the accompanying address. Data and address pair to be sent to the next node is sent through the demultiplexer unit. The multiplexer unit chooses the data according to activity and the priority of the channel, and sends to the processor. The router is designed assuming only one channel sends a packet to the processor at a time, i.e., only one channel is active. However, to prevent temporary conflicts, we assign a priority to each channel. Therefore, in case of a conflict, the packet which comes from a channel that has higher priority is delivered to the processor.

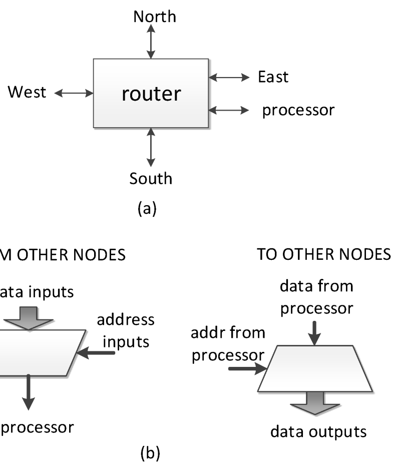

Figure 4. a) Inputs and outputs of the router. Each channel (North, South, etc) has data and address input-output. b) The relations between inputs and outputs of the router.

4. F-CPAMA: Framework for CPAMA

Design of a CPAMA instance is handled by our automation tool F-CPAMA. As shown in Fig. 5, we have developed a set of software codes (Application Program Interfaces, API) and hardware definition codes for F-CPAMA. This tool can be considered as a framework designed to accelerate the work of the user who exploits CPAMA. We also have a Design Space Exploration Tool (DSET) to search the possible best performance of the target CPAMA instance for a set of different configurations. The flow of F-CPAMA is repeated automatically by this tool to search possible best performance by changing configurations, constraints, etc.

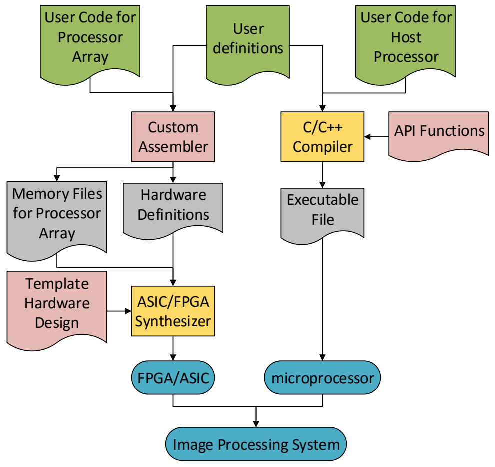

Figure 5. Design flow of a CPAMA instance by using F-CPAMA.

In Fig. 5, the green files are defined by the user. Red blocks are the codes or software provided by F-CPAMA. Yellow blocks are common third-party software. Gray files are hardware or software related files generated by the framework. Blue files are the outputs defining the target architecture.

The User Code for the Host Processor is provided by the user and implements the control of the processor array. Template Hardware Design files of CPAMA are provided along with the framework. The hardware code defining the architecture is vastly generic. Sizes of modules, wires, registers, etc., depend on the User Code for Processor Array and the User Definitions. Custom Assembler (F-Assembler) is the main tool that generates the design files of CPAMA. Parsing the user program and other defined parameters, F-Assembler generates program and constant memory files for each processor. In addition, F-Assembler assigns values to the parameters of the design code.

F-Assembler provides code mapping onto different processors. Therefore, the user can map different functionality to different processors. F-Assembler also provides instruction level parallelism. Register related instruction(s) and an ALU instruction can execute concurrently. F-Assembler generates memory files related to the user program, the files related to the size of the network, and the configurations for the processors. These configurations define the size of the register file and determine instructions that are to be instantiated. Therefore, each processor uses the necessary amount of logic for the defined functionality. For instance, if ADD instruction is not used in a processor program, related ADD logic is not instantiated in the data-path.

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 F-Assembler instantiated all the instructions and registers that are used in those programs in the processor architecture. Then, changing the instruction and constant memory will result in changing the program memory. On an FPGA, run-time programmability of CPAMA can be performed by partially reconfiguring memory blocks. In the ASIC case, the memory contents can be delivered through FIFO and written into the Instruction and Constant Memory. The architectural features related to programmability in ASIC case have not been implemented yet.

The API functions are written in C++ language. These functions are considered to run on the host processor. They are written in pure C++, and are not dependent on a specific processor. Functions are responsible for global control. To mimic the host processor, a SystemC-based testbench environment is designed. HDL design files of CPAMA and API functions written in C++ are simulated together to verify functional correctness.

5. Case Studies

We implemented four different applications on CPAMA to show its performance and scalability. The first application is dot product, which is a core algorithm for many image processing applications. The second one is TIFF2BW [14] application. The third one is 2D Inverse Discrete Cosine Transform (IDCT) application. Finally, the fourth one is block-match which is widely used in video processing.

5.1. Dot Product Application

We have implemented dot product application on a Xilinx Virtex-5 FPGA (xc5vtx240t). This application fits into the second category that is mentioned in Sec. 2, i.e. local. In this algorithm, to compute a result pixel of a specific coordinate, we need the pixels of the input image in the neighborhood of that same coordinate.

We have implemented 86 different configurations to analyze scalability of CPAMA. Table 1 presents a subset of the implemented configurations for $r = 1, 2$ cases. In $r = 2$ case, we only changed the network configuration not the program memory to eliminate the area increase because of the program memory. Since the execution time of synthesis and place & route tool takes too much time for this analysis, we have used DSET. To find the best possible timing result for each processor array configuration, DSET software changes the constraints adaptively and iterates the flow again. However, we had to limit the number of iterations due to long execution times. Consequently, we can say that the area and period results that our software finds are close to the best possible results. As seen in Table 1, CPAMA is scalable and can be configured in various sizes.

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	20227	21811
	4	50	9.290	9.418	19635	22646
	100	2	9.874	9.920	27753	31887

5.2. TIFF2BW Application

To compare our architecture with ADRES [15], TIFF2BW algorithm [14] is implemented. ADRES is chosen as a reference architecture because it is not dependent to a specific device like RAMPSoC [10], and more importantly the TIFF2BW application presented in their study [15] is repeatable. TIFF2BW application fits into the first group (point) of applications that are defined in Sec. 2.

For TIFF2BW application, ADRES instance was configured as a 4×4 array. The precision of the arithmetic-logic operations of ADRES was set to 32-bits. The circuit was implemented using 90nm CMOS technology. For power measurement, they have generated the switching activity using a 1520 by 1496 sample picture from [14]. In dynamic power measurement, off-chip memory accesses are not included. The circuit was simulated at 300MHz.

To make a fair comparison between two architectures, we have implemented a 4×4 CPAMA instance with the same precision that ADRES has. We have implemented the design using a 90nm CMOS library and Cadence tools [16]. To eliminate the effect of inputs in dynamic power measurement, we used the same input image.

Table 2 presents performance comparison of ADRES and CPAMA for TIFF2BW application. CPAMA instance that is compared is the one with a "*". We have added performance values of two more CPAMA instances into the table to demonstrate its scalability and to show performance results of different configurations. As a result, according to our comparison regarding TIFF2BW application, CPAMA consumes 31% less energy, and provides 32% more throughput than ADRES. Moreover, when the #processors of CPAMA increased 4X, the throughput increased almost 4X as well.

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 4x4	300 MHz	303	0.54	NA
CPAMA 4x4*	300 MHz	400	0.37	0.40
CPAMA 4x4	350 MHz	466	0.40	0.42
CPAMA 8x8	333 MHz	1641	0.54	1.45

5.3. IDCT Application

To compare the proposed architecture with another ADRES study [17], we have implemented IDCT as well. IDCT algorithm can be considered in the third group (global) of applications that are defined in Sec. 2. Since the method used for IDCT implementation in ADRES [17] is not stated, we have followed two different methods for implementing IDCT. In the first implementation, we have used Cheng-Wang [18] method. Using this method, 2D-IDCT is implemented by applying 1D-IDCT to each column and row respectively. This method is implemented on a 4×4 CPAMA. In the second implementation, we have used usual matrix multiplication. For matrix multiplication, we have used cross-wired mesh array approach [19]. For the second implementation, CPAMA is configured as a 8×8 array. ASIC implementation is similar with the TIFF2BW application, hence it is not repeated here. Table 3 presents the comparison between CPAMA and ADRES instances for IDCT implementation.

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	90nm LP
ADRES 4×4	312	1.70	19.1	1.08	90nm LP
CPAMA 8×8	303	5.60	21.7	1.41	90nm LP
ADRES 8×8	294	1.65	43.2	NA	90nm LP

5.4. Block-match Application

We have implemented block-match algorithm as a proof-of-concept for multiple frames. Block-match is used in motion estimation. In this application, matching is performed by computing sum of absolute difference (SAD) of two blocks of images. One of these blocks is in the reference frame, the other one is in the current frame. Three different instances of CPAMA, that implement block-match application, are presented in Table 4. Network size of CPAMA is also equal to one block size. Word-length of the processors are set to 16-bits. CPAMA instances are implemented on a Xilinx Virtex-5 FPGA (xc5vtx240t).

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	26575	256

In Table 4, CPAMA 4×4 has a bigger number in throughput, but note that size of the block is smaller than that of CPAMA 8×8 and CPAMA 16×16 configurations.

6. Conclusion

In this paper, a novel architecture is proposed and four different test applications are implemented on the proposed design. The results are compared to that of existing studies. Comparisons suggest that CPAMA provides competitive or better performance in TIFF2BW and IDCT applications, in terms of area occupation, energy consumption and throughput.

CPAMA is a highly configurable architecture that can be used for image/video processing. Especially for the applications that fit into the first and second group (point, local) that are defined in Sec. 2, changing the configuration can be done in a matter of days. Since it is a reusable design, CPAMA improves design and verification time of low-cost, low-power consumer electronics that exploit multimedia tasks.

7. References

[1] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer US, 2010, pp. 449–484.

[2] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in *Field Programmable Logic and App.*, ser. Lecture Notes in Computer Science, P. Y. K. Cheung and G. Constantinides, Eds. Springer, 2003, vol. 2778, pp. 61–70.

[3] C. Jang, J. Kim, J. Lee, H.-S. Kim, D.-H. Yoo, S. Kim, H.-S. Kim, and S. Ryu, "An instruction-scheduling-aware data partitioning technique for coarse-grained reconfigurable architectures," in *Proc. of the SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, New York, 2011, pp. 151–160.

[4] N. R. Miniskar, R. R. Patil, R. N. Gadde, Y. c. R. Cho, S. Kim, and S. H. Lee, "Intra mode power saving methodology for cgra-based reconfigurable processor architectures," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 714–717.

[5] N. Yildiz, E. Cesur, K. Kayaer, V. Tavsanoglu, and M. Alpay, "Architecture of a fully pipelined real-time cellular neural network emulator," *Circuits and Systems I: IEEE Transactions on*, vol. 62, no. 1, pp. 130–138, Jan 2015.

[6] S. Malki and L. Spaanenburg, "A cnn-specific integrated processor," *EURASIP J. on Advances in Signal Proc.*, vol. 2009, pp. 1–14, 2009.

[7] B. Stabernack, K.-I. Wels, and H. Hubert, "A system on a chip architecture of an h.264/avc coprocessor for dvb-h and dmb applications," *Consumer Electronics, IEEE Transactions on*, vol. 53, no. 4, pp. 1529–1536, Nov 2007.

[8] J.-C. Chu, C.-W. Huang, H.-C. Chen, K.-P. Lu, M.-S. Lee, J.-I. Guo, and T.-F. Chen, "Design of customized functional units for the vliw-based multi-threading processor core targeted at multimedia applications," in *IEEE Int. Symp. on Circuits and Sys.*, May 2006, pp. 2389–2392.

[9] S. Pedre, T. KrajnÃk, E. Todorovich, and P. Borensztejn, "Accelerating embedded image processing for real time: a case study," *J. of Real-Time Image Proc*, pp. 1–26, 2013.

[10] D. Göhringer and J. Becker, "High performance reconfigurable multi-processor-based computing on fpgas," in *Parallel Distributed Processing Workshops, IEEE International Symposium on*, April 2010, pp. 1–4.

[11] C. S. Bassoy, H. Manteuffel, and F. Mayer-Lindenberg, "Sharf: An fpga-based customizable processor architecture," in *2009 Int. Conf. on Field Programmable Logic and Applications*, Aug 2009, pp. 516–520.

[12] G. A. Baxes, *Digital Image Processing: Principles and Applications*, 1st ed. USA: Wiley, 1994.

[13] (2017) Fundamentals of image processing. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUDELFT/FIP2_3.pdf

[14] (2017) Mibench: Embedded benchmark suite. [Online]. Available: http://wwweb.eecs.umich.edu/mibench

[15] M. Hartmann, V. Pantazis, T. Vander Aa, M. Berekovic, and C. Hochberger, "Still image processing on coarse-grained reconfigurable array architectures," *J. of Signal Processing Systems*, vol. 60, no. 2, pp. 225–237, 2010.

[16] (2017) Cadence. [Online]. Available: www.cadence.com

[17] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, *Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array*. Springer Berlin Heidelberg, 2008, pp. 66–81.

[18] Chen-Wang, "Inverse two dimensional dct," in *Proceedings of the IEEE ASSP-32*, August 1984, pp. 803–816.

[19] S. Kak. (2017) Efficiency of matrix multiplication on the cross-wired mesh array. [Online]. Available: https://arxiv.org/pdf/1411.3273