HAL open science

Dynamically adaptable architecture for real-time video processing

Nicolas Ngan, Eva Dokladalova, Mohamed Akil, Francois Contou-Carrére

To cite this version:

Nicolas Ngan, Eva Dokladalova, Mohamed Akil, Francois Contou-Carrére. Dynamically adaptable architecture for real-time video processing. IEEE International Symposium on Circuits and Systems (ISCAS'10), 2010, France. 10pp. hal-00622434

HAL Id: hal-00622434

https://hal.science/hal-00622434v1

Submitted on 21 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dynamically adaptable architecture for real-time video processing

Nicolas Ngan*,†, Eva Dokladalova and Mohamed Akil
*Laboratoire Informatique Gaspard Monge, Equipe A3SI
Unite Mixte CNRS-UMLV-ESIEE (UMR 8049)
Emails: {ngann, e.dokladalova, akilm}@esiee.fr

François Contou-Carrère
†Sagem, Massy-Palaiseau
Groupe SAFRAN, France
Email: francois.contou-carrere@sagem.com

***Abstract*— In this paper, we present a new adaptable ring-based architecture for video processing applications. The proposed architecture allows handling pipelined and parallel organization of computation for multiple video flows. A simplified version with four nodes has been implemented on an FPGA for a video application to show the adaptation mechanism between a pipelined and parallel structure.**

I. Introduction

In addition to the quality of image sensors, the modern portable video equipments need a performant embedded computing system enabling more and more of advanced functionalities such as panoramic, picture-in-picture or image fusion applications. Those advanced video processing chains involve the management of multiple video flows depending on the spatial and temporal complexity of the application. Moreover, those complex applications have to be implemented in a variable but limited surface (depending of the portability of the system) and have to meet variable real-time constraint from 25 to 50 frames per second with an image size up to the 1080p HD resolution (1920x1080).

The last decades of research have shown that the performance and flexibility are determined by the balance between its programmability and reconfigurability levels (granularity) of computing systems [1]. In parallel to the highly parallelized architectures, lots of linear architectures has proven that dedicated pipelined architectures [2][3] are the most performant and natural structure for video-based system. However, pipelined architecture depth is always limited by the surface allowed on a chip and the only flexibility remains on the programmability of the processing elements. Furthermore, the size of the pipeline has a direct impact on the latency and the only solutions are to either increase the processing elements efficiency or bypass some processing elements to shorten the datapath which results a loss of computing power.

Despite the numerous dedicated architectures for different image processing problems, there are few systems considering the efficient processing multiple video flows. In general, the processing chains are duplicated or video chains are merged at differents steps. This results a very specific system which excludes any major modification of the structure. Thus, in the context of image sensor evolution (resolution, pixel granularity, video acquisition speed) and as a consequence the video processing algorithms modifications, those rigid solutions could not be applied anymore except at a high development cost between each generation of equipment.

In order to deliver required performances and to offer the flexibility, the interconnections of computing ressources has to be chosen very carefully. In the last few years, ring-based architectures become very trendy in audio [4], graphics [5][6], image such as [7] [8] [9] and general purpose applications [10][11]. These propositions allow to satisfy the performances but, unfortunately, most of those architectures are either unable to manage different parallel input flows or are simply not adapted for embedded systems. Thus, we propose to adapt the datapath for both pipeline and parallel execution according to the video application and the processing element.

In this paper, we present a new adaptable ring-based architecture for video processing applications. The proposed architecture allows handling pipelined and parallel organization of computation for multiple video flows. A simplified version with four nodes has been implemented on an FPGA for a video application to show the adaptation mechanism between a pipelined and parallel structure.

The paper is organized as follows. The Section II presents processing patterns in modern applications and resulting architecture specifications. The architecture proposition is described in Section III and its first FPGA implementation is proposed in Section IV.

II. Specifications for efficient architecture design

From the discussion in the Introduction, we can conclude that design of the efficient computing pipeline is the most important aspect to keep performance but it has to be flexible enough to handle the multiple possible situations such as video loop-back, multiple inputs pipeline execution mode or multiple dependencies. In this paper, we call a pipeline a highly optimized datapath used to connect in a series computing ressources in order to process a given data flow. We say that it is full dedicated solution when the length of used pipeline is minimal and the obtained computing latency is minimal.

The analysis of numerous video processing chain has allowed to identify several repetitive data flow patterns, representing the most frequent pipeline implementations. Let T_i the tasks to be executed, then the different implementations can be illustrated as follows:

Pipeline: it is the most performant and memory cost efficient method. It allows to store and process directly on the flow. Figure 1 illustrates this execution mode. The arrows represent the data dependencies between tasks.

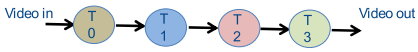

Fig. 1. Basic Video Pipe from Task 0 to Task 3

Multiple pipeline: some video applications need different input flows of the same nature to be combinated such as creating a panoramic with several images or incrust a picture in another (picture-in-picture technique). In this execution mode, multiple flows have to be processed and synchronised according to the tasks (Figure 2).

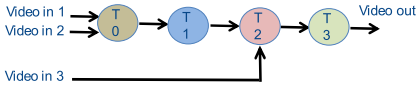

Fig. 2. Video Pipe with multiple input data flows

Multiple pipeline with parallel tasks: this pattern adds new tasks in the video processing chain and those new tasks could be simply chained with the others (increasing the pipeline) or need to work in parallel of the main video processing flow. They can also create new data dependencies as shown in Figure 3.

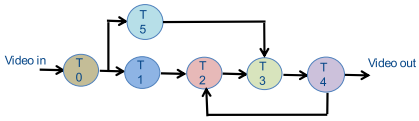

Fig. 3. Video Pipe with a Task 4 and 5

Frame loopbacks execution mode: the video processing chain might require to buffer several image frames.

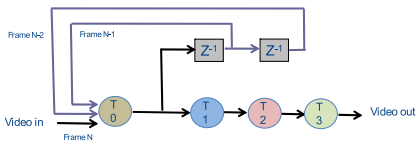

Fig. 4. Video Pipe with frame buffer requests

Let us consider that each task T_i is implemented in a Processing Element PE_i. The main idea of the adaptation is to morph dynamically between the different data flow patterns (Fig. 1-4). The principle is also partially shown in Figure 5. For multi-tasking purposes, this adaptation should load or unload the pipeline from PEs dynamically and use the unloaded ones to realize parallel processing on different data flows.

Those adaptations can only be done physically by adding some additionnal dedicated hardwares (called $DFRs$ i.e. *DataFlow Routers*) to create the different connections allowed between PEs.

(a) Pipeline (b) Parallel tasks

Fig. 5. Datapath modification requirement examples

We propose to interface the PEs with DataFlow Routers ($DFRs$) which are interconnected on a traditional ring structure. All datapath modifications are realized inside the DataFlow Routers. We call $DFR_{I/O}$ a DFR node equipped with external input and output video flows. The ring topology has the advantage to be relatively simple to implement and each node has only and always two neigbours which means that routers and interconnections can be implemented at a limited cost [12] compared to much more complex mesh structures. The structure can be used in a standard pipeline style computation. In addition, in the case of high area constraint, the ring topology facilitates the modifications of PEs (internal parallelization strategy or increasing the working frequency to keep the video rate).

III. Architecture Presentation

The proposed architecture is described in Figure 6. It is composed of DFR and $DFR_{I/O}$ nodes linked together forming a ring topology. Note that N_{DFR} and $N_{DFR_{I/O}}$ represent the number of each DFR nodes type. To be able to interface directly K video sources, we use $K+1$ rings: K unidirectionnal data flow rings (main video channels) and one ring in the inverse direction to synchronize the flow and to communicate some information or processing elements states.

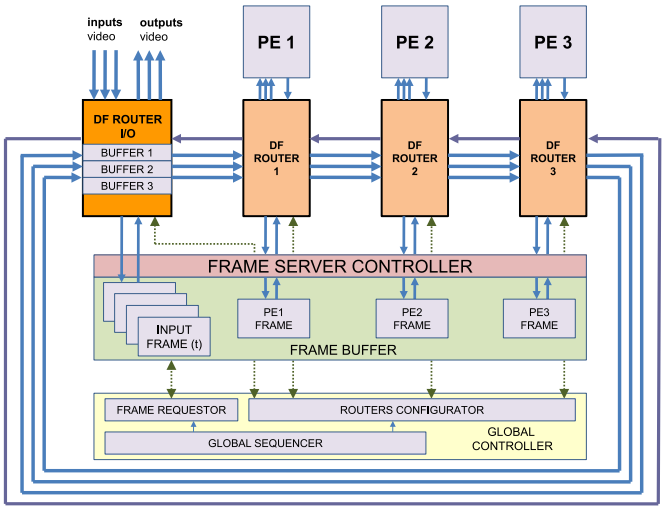

Fig. 6. Global architecture: $N_{DFR} = 3$, $N_{DFR_{I/O}} = 1$ and $K = 3$

A. Architecture description

1) DataFlow Router: A DataFlow Router (Figure 7) is in charge of redirecting all the video data flows required for its

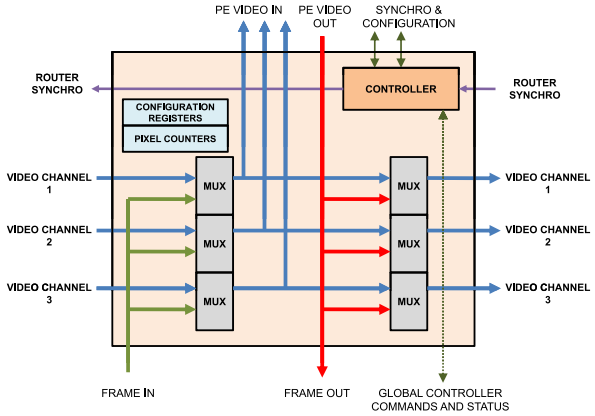

Fig. 7. DataFlow Router internal structure: $K = 3$

connected PE. Its main actions are switching input data flows from video channels, PE and memory. The router contains buffers to synchronize input flows, mutiplexers, configuration registers and pixel counters to track the status of the processing element video output. It receives a command from the global controller containing its routing configuration and potential PE modification information. The DataFlow Router I/O ($DFR_{I/O}$) has a similar structure but is dedicated to video inputs-outputs and contains line buffers for each video channel.

2) Frame Buffer Server: The Frame Buffer Server stores all the frames needed for the video processing chain and results from PEs. It is a shared memory composed by several buffer memory slots dedicated to each PEs and those slots could be exchanged to share frame results. A frame buffer server controller manages all the frame requests from the global controller for the PEs.

3) Global controller: The global controller contains the video processing chain sequence and all the router adaptation required. Its role is to configure all the routers in the architecture and make all the necessary memory request to the frame server to give the right video flow when needed. At the beginning, the controller configures every router for each video channel (parallel or pipeline mode) and the $DFR_{I/O}$ is configured for video loopbacks or not. It can request video flows from the frame buffer server which capture permanently the input videos. The controller can decide to unload the pipeline from one processing element to do some parallel tasks for the following frames. It dynamically configure the DataFlow Router for a specified video channel. The video channel bypasses the router and the processing element can work transparently on a video data flow from memory. The DFR_i reports to the controller that the PE_i task is done when the processed pixel counters for the processing element is reached. Let T_{COM} the time for the controller to sets the DFR configuration registers, the processing element task and the quantity of pixels to process (a complete frame or frame part) by setting the DFR pixel counters. T_{DFR} is the DFR reconfiguration time and T_{PE} the processing element one. The total adaptation time T_{ad} for a DFR_i is the sum of those times (1).

$$T_{ad}(i) = T_{COM}(i) + T_{DFR}(i) + T_{PE}(i) \quad (1)$$

4) Processing Elements: Each PE can accept up to three different video flows to be combinated and it outputs the processed video flow. In this paper, PEs can be considered as black boxes and all the local memory managements is done inside them according to the input video flows from different sources.

IV. FPGA IMPLEMENTATION

We propose to implement a video application illustrated by Figure 8 combining several pipelined image filters. It inputs two video flows that are combinated in T_3. We will then compare the performance of our solution compared to a full dedicated one. To implement in a three-nodes solution for instance, we divide the application in two parts: a pipeline part and a parallel part. Those two computing parts will be temporally multiplexed in the ring and our structure has to be dynamically adapted to realize the different datapaths.

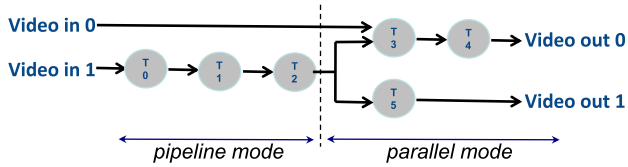

Fig. 8. Video application example

Obviously, making a loop with less PEs will have an impact in the global latency unless we can increase the frequency. Our goal is to show that with only three PEs, it is possible to implement the application above by one video loopback and a pipeline-parallel adaptation with our proposed structure.

A. Implementation

This design has been implemented on an Altera Stratix III EP3SL150 FPGA (150K logic elements) [13]. To test our proposition, we prototype the architecture with three DFR nodes ($N_{DFR} = 3$), one $DFR_{I/O}$ ($N_{DFR_{I/O}} = 1$) and three main video channels ($K = 3$) as illustrated in Figure 6 and 7.

B. Surface and Performance results

Let S_{DFR} and $S_{DFR_{I/O}}$ the surface occupied by DFR and $DFR_{I/O}$ respectively. S_{ctrl} is the surface of the controllers and the communication links. The surface overhead cost S_{cost} compared to a full dedicated implementation is given in Equation (2).

$$S_{cost} = \sum_{i=1}^{N_{DFR}} S_{DFR}(i) + \sum_{j=1}^{N_{DFR_{i/o}}} S_{DFR_{i/o}}(j) + S_{ctrl} \quad (2)$$

Results in FPGA surface are shown in Table I and S_{cost} represents 762 Logic Elements (LEs) for our implementation.

Similar PEs of 1452 LEs are used and a global surface comparison is given in Table II. Let Lat_{DFR} the latency

TABLE I

FPGA SURFACE RESULTS FOR ROUTING AND CONTROL ELEMENTS (ALTERA STRATIX III EP3SL150)

	S_{DFR}	$S_{DFR_{I/O}}$	S_{ctrl}
Surface (LEs)	122	181	215

TABLE II

COMPARISON OF TOTAL ESTIMATED SURFACE ON FPGA (ALTERA STRATIX III EP3SL150)

	Full dedicated	Adaptable
Surface (LEs)	8815	5124

between two neighbours PEs across the routers and $Lat_{PE}(i)$ the latency for PE_i. For our implementation, the PEs are dedicated structure that works on the flow and changing the functionality of a PE is simply done by switching coefficients ($T_{PE}(i) = 1$). We also use PEs which have the same latency ($Lat_{PE}(i) = 18$ clock cycles). The global controller is dedicated to the structure with specific input and output ports and finite state machines. All the DFRs have a direct connection on the global controller ($T_{COM}(i) = 3$) and the DFR modifications are done by multiplexers controlled by registers ($T_{DFR}(i) = 2$ and $Lat_{DFR} = 2$). Consequently, $T_{ad} = 6$ clock cycles for our implementation.

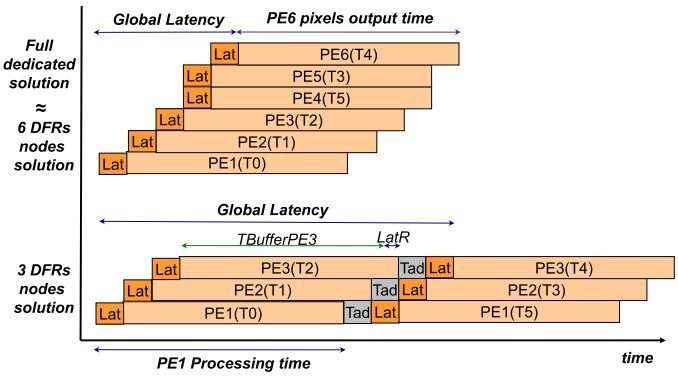

Fig. 9. Latency comparison between architecture solutions

Figure 9 shows the pixel outputs of each PEs for a burst of pixels from a 1080p image ($burst\ size = 1920$). We can notice the processing time and the latency comparison between a full dedicated solution and our adaptable solution. We can see in this figure that Lat_{DFR} is negligible or even hidden by PEs latencies. Note that our solution needs an additionnal buffering (during $T_{BufferPE3}$) in $DFR_{I/O}$ to realize the data loopback. We measured a latency of 1998 clock cycles (it is related to $burst\ size$ value) with three DFR nodes. However, by keeping the same structure and using six DFR nodes, we have similar performances in latency compared to the full dedicated one as shown in Figure 9 because Lat_{DFR} is negligible. Note that $Video\ in\ 0$ from Figure 8, is directly stored in the frame buffer and injected when needed on the right node for T_3 contrary to a full dedicated structure with a fixed size buffer for synchronization.

From this implementation, we can conclude three things: First, for a given application, we could use the right number of nodes to get performance closed to a full dedicated structure and when needed, by keeping the same structure, our system can work with less number of nodes by making several video loopbacks thanks to the ring structure and the pipeline-parallel adaptation mechanism. Thus, the remaining nodes can work on others tasks in parallel. Secondly, our system can work on different parallel video flows and the PEs can work transparently without knowing the source either from a PE or the frame server which is important for design reuse. Finally, depending on the PE complexity in time and surface, we show that the cost in surface for implementing our proposition is relatively interesting for gaining such flexibility on datapath and performance.

V. CONCLUSION

In this paper, we have presented an adaptable ring-based architecture proposition that could handle both pipeline and parallel processing in the same application for an input video flow. A simplified version with four nodes has been designed on an FPGA for a video application with two video flow inputs showing the parallel-pipeline adaptation mechanisms. Future works will focus on the study of scalability of the structure. Consumption aspects will also be analyzed.

REFERENCES

[1] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 642–649.

[2] C. Clienti, S. Beucher, and M. Bilodeau, "A system on chip dedicated to pipeline neighborhood processing for mathematical morphology," 2008.

[3] E. Dokládalova, R. Schmit, S. Pajaniradja, and S. Amadori, "Carvision: SOC architecture for dynamic vision systems from image capture to high level image processing," in *MEDEA DAC*, 2006, (4 pp.).

[4] Creative, "X-fi ring architecture, www.creative.com."

[5] ATI, "Radeon x1800 memory controller - whitepaper," Tech. Rep., 2005.

[6] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, and M. Abrash, "Larrabee: A many-core x86 architecture for visual computing," *SIGGRAPH*, 2008.

[7] J. van der Horst, R. van Leeuwen, H. Broers, R. Kleihorst, and P. Jonker, "A real-time stereo smartcam, using fpga, simd and vliw," 2006.

[8] N. Farrugia, F. Mamalet, S. Roux, F. Yang, and M. Paindavoine, "Design of a real-time face detection parallel architecture using high-level synthesis," *EURASIP Journal on Embedded Systems*, 2009.

[9] G. Sassatelli, P. Benoit, L. Torres, G. Cambon, J. Galy, M. Robert, and C. Diou, "Systolic ring: A new approach for dynamical reconfigurable architectures," in *GRETSI*, 2002.

[10] T. Collette, C. Gamrat, D. Juvin, L. L. Jean-Franois Larue, M. Peythieu, R. Schmit, and M. Viala, "Symphonie massively parallel computer : Modelling and design," *Traitement du Signal*, vol. 14, 1997.

[11] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network : Built for speed," *IEEE MICRO*, 2006.

[12] M. Saldana, L. Shannon, and P. Chow, "The routability of multiprocessor network topologies in fpgas," 2006.

[13] Altera, "Stratix iii - user handbook, www.altera.com," Tech. Rep., 2008.