A HIERARCHICAL IMPLEMENTATION OF HADAMARD TRANSFORM USING RVC-CAL DATAFLOW PROGRAMMING AND DYNAMIC PARTIAL RECONFIGURATION

Manel Hentati, Yassine Aoudni, Jean François Nezan, Mohamed Abid

A HIERARCHICAL IMPLEMENTATION OF HADAMARD TRANSFORM USING RVC-CAL DATAFLOW PROGRAMMING AND DYNAMIC PARTIAL RECONFIGURATION

Manel Hentati[1,2] *, Yassine Aoudni* [2] *, Jean-François Nezan*[1] *and Mohamed Abid* [2]

[1] INSA/IETR, 20, av des Buttes de coesmes CS 14315 F-35043 RENNES, France
[2] ENIS/CES, Route Soukra B.P. 1173, 3038 Sfax, Tunisie

ABSTRACT

This paper presents an efficient design method used to implement a hierarchical architecture of Hadamard transform module. The proposed design method is based on the use of RVC-CAL dataflow approach and dynamic partial reconfiguration technique (DPR). The DPR technique allows reconfiguring a part of the FPGA area with different functionalities at runtime. It is a promising solution to increase performance in the system. RVC-CAL is a specific language for writing dataflow models which is introduced by MPEG-RVC video standard. RVC-CAL description is composed of a set of interconnected blocks (actors). Several dataflow models of the same application can be used in the design process. In this work, the hierarchical architecture of Hadamard module is composed of three levels. And each one contains a set of blocks. The DPR is applied between these blocks to switch from level to another. To achieve this implementation, in the first, the Hadamard blocks are described in RVC-CAL language and a specific RVC-CAL tool is used to generate automatically their hardware description. Then, the DPR design flow is applied. In our design method, we use xilinx tools and Virtex-5 FPGA board. To evaluate our implementation, we compare its with two other architectures in terms of area occupation, power consumption and execution time.

Index Terms— Dynamic partial reconfiguration, Hadamard transform, RVC-CAL, hierarchical architecture, design approach, execution time, FPGA.

1. INTRODUCTION

Modern applications like audio/video compression, image processing and 3D graphics, need high performance and efficient architectures to be executed, especially in the embedded systems domain. These applications often include a video standard such as MPEG-2, H263, etc, which suffers from reusability and generality of the code (usually describe on a C/C++ monolithic). To overcome these limits, Moving Picture Expert Group (MPEG) proposes a new standard called MPEG-RVC [1] which allows dynamic development and implementation of existing or new video coding solutions. In this ways, MPEG-RVC improves the flexibility and reutilisabilty of codec features and facilites the support of various codec. MPEG-RVC introduces a dataflow language called RVC-CAL to describe the different functions of codec as a networks blocks named actors. This description can be easily translated on a specific language: software or hardware depending on the target architecture. This translation is supported by a set of tools defined by MPEG-RVC including OpenDF [10] for the simulation, CAL2HDL [7] [6] and ORCC [9] for the automatic code generation (C, HDL, LLVM, ...).

In this paper, we aim to design a hierarchical implementation of Hadamard transform in RVC framework. Indeed, Hadamard architecture is composed of three level and each level includes four blocks, which are executed separately and in parallel. We propose to use the dynamic partial reconfiguration (DPR) to commute from level to another.

The DPR of FPGA is an attractive feature which allows changing some processes of an FPGA device while other processes continue in the rest of the device. Xilinx has supported partial reconfiguration for many generations of devices (Virtex-II, Virtex-4, Virtex-5). DPR offers countless benefits across various researches works. In fact, DPR allows the improvement of FPGA area efficiency and the decrease of power consumption [2] [19]. In addition, DPR seem to be a promising solution to design flexible and adaptive system in the RVC framework. The proposed design method can be easily extended for the wide diversity of RVC applications. In the remainder of this paper, In section 2, we present the RVC framework. Section 3 explains the specificities of Hadamard transform. Section 4 presents an overview on dynamic partial reconfiguration. Section 5 exposes the design approach to implement Hadamard module using the DPR technique. The experimental results are reported in Section 6. Section 7 concludes the paper and discusses some future directions.

2. RVC FRAMEWORK

The standard MPEG-RVC is developed by MPEG. It aims at providing a unified high-level specification of current and future MPEG video coding technologies by using dataflow models. This standard offers the means to overcome the lack of interpretability between the many video codecs deployed

in the market. MPEG-RVC is composed of MPEG-B and MPEG-C standards. The MPEG-C standard [18] presents the library of video coding tools (VTL) employed in existing MPEG standards. And the MPEG-B standard [4] presents the overall framework and the standard languages used to describe the different components of the framework. An abstract decoder is built as a block diagram in which blocks define processing entities called Functional Units (FUs) and connections represent the data path. These FUs are described by high level language called RVC-CAL [3] which is a textual and domain specific language for writing dataflow models. The main advantage of using this language is that, it is

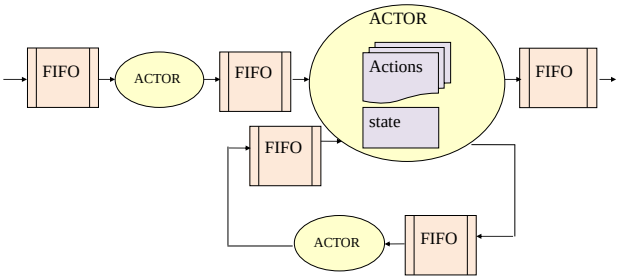

Fig. 1. RVC-CAL Dataflow model.

placed between the C and the VHDL languages and there are open source tools have been developed to automate the transformation of the RVC-CAL code to C (CAL2C [5], ORCC) and to VHDL/VERILOG (CAL2HDL, ORCC). RVC-CAL describes algorithms by using a set of encapsulated dataflow components called actors (FUs) (figure 1). Actors are completely independent from each others and they can communicate by exchanging packet data (called tokens) along FIFOs channels. The topology of the connections between actors input and output ports constituted a network, which is expressed by using an XML dialect known as network language (XDF). The behavior of the actor is a set of actions and only one action is active at a time. Indeed, an actor can read inputs and messages (tokens), modify internal state and produce outputs (tokens). These actions depend on some constraints such as the priority of each action, the state of the actor, the messages availability and the input values (tokens).

3. HADAMARD TRANSFORM

This section introduces the Hadamard transform mechanism and our proposed architecture.
The Hadamard transform derives from a generalized class of the Fourier transform. It consists of a multiplication of a $2^m \times 2^m$ matrix by an Hadamard matrix (H_m) that has the same size. Here are examples of Hadamard matrices. H_0 is the identity matrix so $H_0 = 1$. For any $m > 0$, H_m is then deducted recursively by (1).

$$H_m = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix} \quad (1)$$

The Hadamard is a transform used in many image and video coders such as the LAR (Locally Adaptive Resolution) [12]. There are different Hadamard modules (H1, H2, ...) which are executed according the block size image. It means that, if we have 2×2 block size image, we will use Hadamard (H1). The H2 is exploited, if we have 4×4 block size image.
In this paper, we adopted the (H2) Hadamard transform which has 16 inputs and produces 16 outputs. Many architectures are possible to achieve the H2. We propose the architecture which is presented in figure 2.
This architecture is composed of three levels. And each level

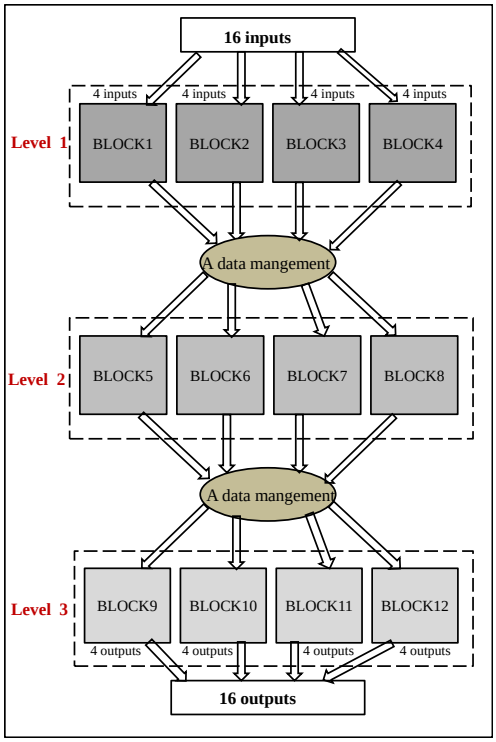

Fig. 2. Hierarchical architecture of Hadamard transform (H2).

contains four blocks. Each block has four inputs and four outputs. It contains a set of arithmetical functions. Blocks which are included in the same level are executed separately and in parallel. All blocks are implemented in hardware. Two data management modules are used to save the intermediate results and to transfer the adequate inputs values to blocks. These two modules are implemented in software.

4. DYNAMIC PARTIAL RECONFIGURATION

Xilinx is currently the only major FPGA vendor to offer support for DPR in their programmable logic devices. In this paper, we focus on these devices and we provide some background on partial reconfiguration technology. Then, we discuss works that exploit this technique to design a dynamic and partial reconfigurable system on chip.

4.1. Background

Dynamic partial reconfiguration is the ability to change the configuration of part of an FPGA device while other processes continue in the rest of the device. A partial reconfigurable design typically comprises an area for static modules and one or more partial reconfigurable region for partial modules. The static modules contain logic that will remain constant during partial reconfiguration. Partial reconfiguration module is the design module that can be swapped in the device on the real-time, multiple modules can be defined for a specific FPGA region. In the early Virtex family devices like Virtex-II and Virtex-II Pro devices, we must partially reconfigure whole columns. Recently, in Virtex-4/5/6 devices, the partial reconfigurable region (PRR) is rectangular of arbitrary size and may be located anywhere with no overlapping. The figure 3 illustrates an example of static and dynamic part for a reconfigurable system.

During partial reconfiguration process, the routing signals

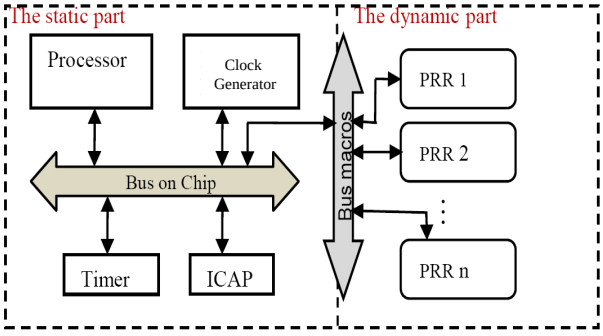

Fig. 3. Static and dynamic part for a reconfigurable system.

used for inter-modules communication should be unchanged when the module is reconfigured. This fixed routing bridge of communication is achieved by using bus macros. Bus macros are hard macros, which is located at the edge of boundaries separating dynamic and static regions.

To load the partial bitstreams, designer should use a reconfiguration controller and an internal configuration access port (ICAP). The Virtex-II series are the first architectures that support ICAP. This latter is a subset of the SelectMAP interface having fewer signals. The ICAP [13] only deals with partial configurations and does not have to support different configuration modes. It gives an 8 input/output bits data bus. While, with the Virtex-4 and Virtex-5 series, the ICAP interface has been updated with 32 input/output bits data bus to increase its bandwidth and speed up the reconfiguration time. The intuitive benefits of using DPR are: the improvement of FPGA area efficiency, the augmentation of architectures flexibility, the decrease of power consumption [17] and smaller FPGAs can be used to run an application because to commute between different modules, we can just load the partial reconfiguration bitstream. However, the implementation process for DPR is still complex and tedious. Therefore, the designer should have a thorough understanding of the underlying device and design methodology. Moreover, PR latency is one of the most critical aspects in the implementation of DPR system, if it is not brief enough, the PR interest to build efficient systems can be jeopardized [11]. In our case study, the partial reconfigurable modules are blocks, which are included in Hadamard transform (H2).

4.2. Related works

With the appearance of dynamic partial reconfigurable FPGA, many researchers are interested to explore the DPR in their flow design, in order to enhance their system performance. As results various methodologies have been developed.

Initially, Xilinx proposes a design flow called Modular Design [24]. In this method, the dynamically reconfigurable modules must occupy the whole height of the device and their positions are fixed (as specified in the top-level design). The interconnections between the modules are fixed also. Therefore, the modification of module position requires the generation of new partial bitstreams, even if the implementations do not actually modify. This methodology is based on using Virtex-II and Vitrex-II Pro FPGA architectures. Lysaght, et al [14] present a DPR design methodology using Virtex-4 FPGA. This approach is based on the early access partial Reconfiguration flow (EAPR) [15] introduced by Xilinx. In this design flow, the static and reconfigurable parts are implemented separately. Then, there are merged. The reconfiguration region is rectangular of arbitrary size and may be located anywhere with no overlapping. The bus macros is used, to implement the communication ports to interface static and dynamic regions of a design. iMPACT Xilinx tool for downloading bitstreams to program devices, is exploited to download partial bitstreams via JTAG interface. The limitations of this approach are the partial bitstreams for a module to be executed on a reconfigurable region must be predetermined and we can only use Xilinx tools version 9.2. Hübner et al. [25] propose a slot based architecture with the novelty that they can give a high level specification of the setup and all the communication macros are automatically placed. Each module connects to the On-chip Peripheral Bus (OPB) via the communication macros. The tool flow generates the entire

bitstreams in EDK and then uses JBITS to cut out the partial bitstreams. JBits is a tool provided by Xilinx that uses Java classes to represent the bitstream and has functions able to change the bitstream at runtime. JBITS can only support the Virtex-II family. Horta et al. [23] present a design flow based on the use PARBIT tool to reconfigure partial bitstreams in Virtex-E. This tool allows transforming and restructuring bitfiles created by standard Computer Aided Design tools into partial bitstreams that program Dynamic Hardware Plugging (DHP).
All the above-proposed design methods are depended on some constraints such as the FPGAs family, tools, etc. These constraints have made the adaptation of these methodologies in the RVC context using virtex-5 technology is complex. Therefore, to propose a new methodology ensures high performance and easy adaptation of RVC application requirements is important. In this paper, we propose a new solution to implement RVC application using DPR in virtex-5 FPGA. We exploit Xilinx tools in version 12.3. Our solution, is flexible and can be adapted on full RVC applications.

5. DESIGN APPROACH

The main objective of this work is to exploit DPR technique to design a hierarchical implementation of Hadamard module. We use the system architecture shown in figure 4.
This architecture is composed of:

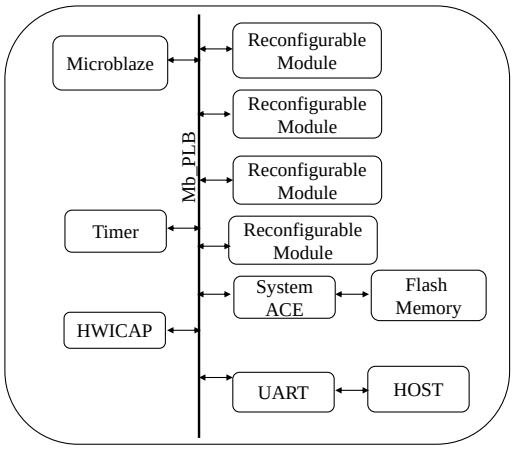

Fig. 4. An embedded architecture for the design method.

- A Microblaze, which is a 32-bit RISC soft processor core [20] to manage the configuration of different blocks.
- An UART provides serial communication between PC and FPGA.
- A ICAP core has been utilized which run at 100 MHz and has 4 bytes width port. So by using this will be able to reach 400 megabytes per second (MB/s).
- A flash compact memory to store the partial bitstreams.
- A timer to measure the execution time.
- Four reconfigurables modules.
- A bus PLB to connect all the peripherals.

In our implementation, we have four reconfiguration regions and twelve reconfigurable modules. In the first, the four blocks of the first level are reconfigured. They are executed in parallel and separately. The results provided by these blocks are saved. Then, these four blocks are removed and replaced by the four blocks of the second level. They use results of the first level as inputs and produced intermediate results, which will be used by blocks from third level and provide the final results.
A data management is used in this design which is a C code. It is executed by the Microblaze processor. This code allows managing efficiently the partial reconfiguration, transferring data and saving intermediate results. To achieve the implementation, we should take into account these three constraints [16]:

- Reconfigurable modules implemented on the same place, must have the same inputs/outputs.
- The area of the reconfigurable partition region is at least equal to the greater of two area needed for each of two reconfigurable modules.
- A unit of control is required to manage efficiently the partial reconfiguration. This unit control is a code written in C and executed by a Microblaze processor.

The design flow proposed in this work is based on using Xilinx FPGA device with dynamic partial reconfiguration (DPR) technique. Our approach comprises a set of steps, which are necessary to implement RVC applications using DPR. During the initial phase, the static modules and the partial reconfiguration modules (PRM) are described in HDL language. The static modules are developed manually. However, the partial reconfiguration modules are described in RVC-CAL language. And, they are automatically transformed in hardware description using CAL2HDL tool. The generated code is formed by VERILOG files that present the actors and a VHDL file for the top. The top file defines the highest hierarchical representation of the design connections. The connection between the actors is insured by synchronous or asynchronous FIFO buffers. The PRMs are, then, synthesized using Xilinx ISE, a tool that compiles HDL code and generates the corresponding netlists for each module. In the next step, Xilinx Platform Studio (XPS) is used to design an embedded System-on-Chip (SoC) with the static logic, PRMs and other peripherals required to build a complete DPR solution. The XPS tool is exploited also to specify a software code that will run on the MicroBlaze. This code will be compiled to generate binary file. In the step follow, the system

netlist and constraints specification from XPS is fed into Xilinx PlanAhead, a tool used to perform placement and routing and generation the full and partial reconfiguration bitstream. Finally, we use again XPS to merges the full bitstream from Xilinx PlanAhead with the compiler software to generate a final downloadable bitstream, called the System ACE file. The System ACE file is copied onto the compact flash card and the card is plugged into the FPGA to bring up the design on the next power cycle.

6. RESULTS

In order to evaluate our design, we compare it with two other architectures. The first one includes one block (actor). And the second one is pipelined architecture. We have employed a set of Xilinx tools version 12.3 [21]: the ISE tool for the synthesis, the PlanAhead visual floorplanning tool for iterative design and placement, and EDK tool for creating and implementing the project.
Figure 5 presents the hardware resources requirements of these three architectures.

According figure 5, the hierarchical architecture uses less

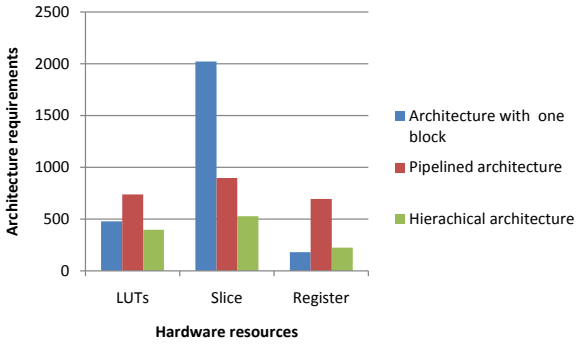

Fig. 5. On-chip area requirements.

area than the two other ones. Because it implements only the needed blocks. However, architecture with one block and pipelined architecture require more hardware resources to implement simultaneously the entire Hadamard application. In figure 6, we give an accurate estimation in terms of total power consumption for the above three architectures. To get an available estimation of power consumption, we used Xpower Xilinx tool [22]. The approach used by this tool consists of providing information including the number of LUTs, the number of flip-flops, etc and the clock frequency to determine the power consumed by the FPGA for a given temperature.

Total power consumption of hierarchical architecture is less as comparing to the two other architectures. We can conclude

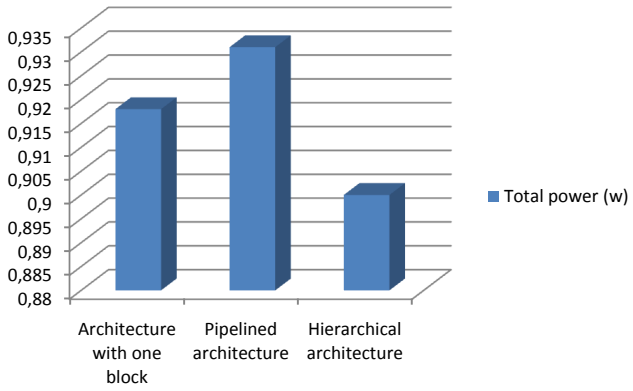

Fig. 6. Power consumption estimation.

that by using DPR, the designer is now able to save area and power.
Figure 7 gives the experimental results in term of execution time in millisecond while varying the number of inputs data for the hierarchical architecture, the architecture with one block and the pipelined architecture. We should note that, the execution time of hierarchical architecture is composed of the required time to achieve computation and the sum of the configuration time of the blocks. The time configuration is the time required to download the configuration data before the system is ready to execute. Table 1 shows the partial bitstreams size and the configuration time. We note that, the blocks which are placed in the same reconfigurable region (RR), have the same bitstreams size. The configuration time can be calculated using the equation 2.

$$t = \frac{bitstream * 8}{32 * 100} \quad (2)$$

The configuration time depends on the bitstreams size and

Table 1. Configuration time.

	Size per bitstream (Bytes)	Configuration time (μs)
Blocks (1, 5, 9) placed in RR1	48128	120,32
Blocks (2, 6, 10) placed in RR2	46080	115,20
Blocks (3, 7, 11) placed in RR3	43008	107,52
Blocks (4, 8, 12) placed in RR4	45056	112,64

the ICAP performance. Indeed, by using ICAP of virtex-5 we will be able to reach 400 megabytes per second (MB/s). However, the ICAP of Virtex-II allows achieving only 50

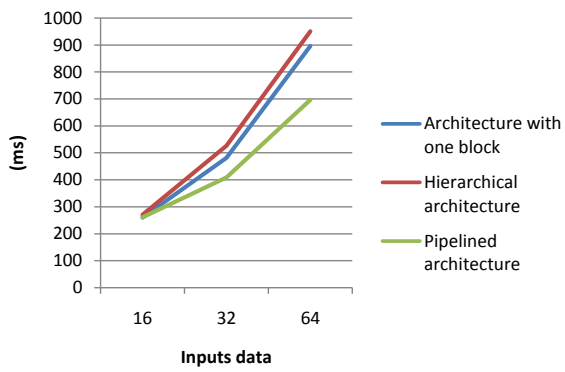

Fig. 7. The execution time.

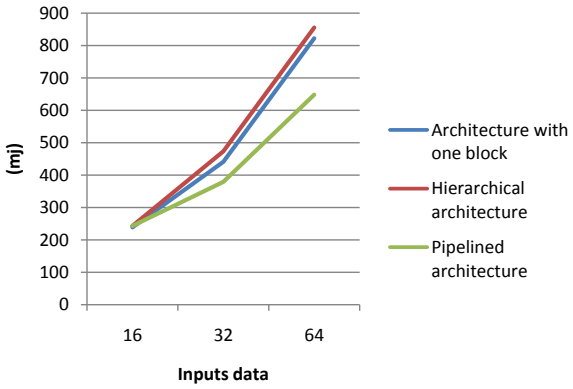

Fig. 8. The energy consumption.

megabytes per second (MB/s).
The comparison between the execution time and the configuration time shows that, the execution time is very higher than configuration time. So, the configuration time doesn't degrade the execution time of the application.
In spite of the limited configuration time, the execution time of hierarchical architecture is higher compared to two others architectures. This is may be due to use of data management module to save and transfer data. While, the two other architectures don't require this module.
Figure 8 presents the energy consumption for the three proposed architectures. The energy consumption is calculated using the equation 3.

$$Energy = total\ power * execution\ time \quad (3)$$

The energy consumption of the pipelined architecture is lower as regards to hierarchical architecture and architecture with one block.
The comparison between these three architectures indicates that, the pipelined architecture is more performance in terms execution time and energy consumption. While, the hierarchical architecture allows saving power consumption and area utilization.

7. CONCLUSION

This paper deals with the hierarchical implementation of Hadamard transform (H2) using Xilinx FPGA device with dynamic partial reconfiguration (DPR) technique. Hadamard module is composed of three levels and in each level contains four blocks. The DPR is applied between these blocks to switch from level to another. A data management module is used to control the reconfiguration process, to save intermediates results and to transfer data between blocks. This module is a C code executed by the processor. The implementation results demonstrate that, the dynamic and partial reconfiguration design approach is an effective way to reduce the area utilization and the power consumption. However, the execution time is higher as compared two other architectures. Therefore, the choice of architecture is based on required performance (less execution time or area utilization).
In this paper, our application is described in RVC-CAL. Among the advantages of this language is the ability to describe easily the same application in different architectures. We have chosen manually a hierarchical architecture for Hadamard module which is composed of three levels. But, it is important to study the impact of the tasks (blocks) distribution on the system performance to find the best architecture. Future work, will aim to automatically define the distribution of tasks from networks actors. We will develop a scheduler algorithm to define the best architecture to execute tasks that forming the RVC application. This algorithm must take into account the dependency between tasks and tasks priority. By using such algorithm, we can enhance application performance in the MPEG-RVC framework. We plan also to integrate this algorithm into the RVC-CAL tools in order to automatically implement RVC applications on FPGAs using the DPR functionality.

8. REFERENCES

[1] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, *Overview of the MPEG reconfigurable video coding framework*, Journal of Signal Processing Systems, 2009.

[2] P.Manet, D.Maufroid, L.Tosi, G.Gailliard, O.Mulertt, M.DiCiano, J.-D. Legat,D.Aulagnier, C.Gamrat, R.Liberati, V.LaBarba, P.Cuvelier, B.Rousseau,

P.Gelineau, *An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications*, EURASIP Journal on Embedded Systems 2008.

[3] M. Mattavelli, I. Amer, and M. Raulet, *The Reconfigurable Video Coding Standard [Standards in a Nutshell],* Signal Processing Magazine, IEEE, vol. 27, no. 3, pp. 159 -167, May 2010.

[4] MPEG Systems Technologies Part 4: Codec Configuration Representation, ISO/IEC FDIS 23001-4, 2009.

[5] Ghislain Roquier, Matthieu Wipliez, Mickaël Raulet, Jörn W. Janneck, Ian D. Miller, and David B. Parlour. *Automatic software synthesis of dataflow program : an MPEG4 Simple Profile decoder case study*, In IEEE-Workshop on Signal Processing Systems (SiPS 2008), Washington, D.C., USA, pages 281-286, 2008.

[6] R. Gu, J. W. Janneck, S. S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker, *Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis*,IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 11, pp. 16461657, 2009.

[7] Cal2HDL-openforge source Available from: http://openforge.sourceforge.net. [Accessed: December 2010].

[8] Jörn W. Janneck, Ian D. Miller, David B. Parlour, Ghislain Roquier, Matthieu Wipliez, and Mickaël Raulet, *Synthesizing hardware from dataflow programs: An MPEG-4 simple profile decoder case study* in Proceedings of IEEE Workshop on Signal Processing Systems, SiPS 2008.

[9] J. W. Janneck, M. Mattavelli, M. Raulet, and M. Wipliez, *Reconfigurable video coding: a stream programming approach to the specification of new video coding standards*,in MMSys 10: Proceedings of the first annual ACM SIGMM conference on Multimedia systems. New York, USA: ACM, pp. 223234, 2010.

[10] S. Bhattacharyya, G. Brebner, J. Eker, J. Janneck, M. Mattavelli, C. von Platen, and M. Raulet, *OpenDF - A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems,First Swedish Workshop on Multi- Core Computing*, MCC , Ronneby, Sweden, November27-28, 2008.

[11] Pierre Bomel, Jeremie Crenne, Linfeng Ye, Jean-Philippe Diguet, and Guy Gogniat,*Ultra-Fast Downloading of Partial Bitstreams through Ethernet*, ARCS 2009, LNCS 5455, pp. 72-83, 2009.

[12] O. Deforges, M. Babel, L. Bedat, and J. Ronsin, *Color LAR Codec: A Color Image Representation and Compression Scheme Based on Local Resolution Adjustment and Self-Extracting Region Representation* , IEEE Trans. Circuits Syst. Video Techn., vol. 17, no. 8, pp. 974-987, 2007.

[13] Simen Gimle Hansen, Dirk Koch, Jim Torresen *High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro*, IPDPS Workshops: 174-180, 2011.

[14] P. Lysaght, B. Blodget, J. Mason, B. Bridgford, and J. Young, *Enhanced Architectures, Design Methodologies and CAD Tools for dynamic reconfiguration of XILINX FPGAs*,in 16th Int. Conf. on Field Programmable Logic and Applications (FPL2006), pp. 12-17, 2006 .

[15] *Early Access Partial Reconfiguration*, User Guide for ISE 9.2.04i September 2008.

[16] Manel Hentati, Yassine Aoundi , Jean François Nezan, Mohamed Abid, Olivier deforges *FPGA dynamic reconfiguration using the RVC technology: Inverse Quantization case study*, *Conference on Design and Architectures for Signal and Image Processing (DASIP), Finland ,2011.*

[17] B. Krill, A. Ahmad, A. Amira, and H. Rabah, An efficient FPGA-based dynamic partial reconfiguration design flow and environment for image and signal processing IP cores*, Journal of Signal Proc.: Image Comm., vol. 25(5), pp. 377-387, 2010.*

[18] MPEG Video Technologies Part 4: Video Tool Library, ISO/IEC FDIS 23002-4, 2009.

[19] Cindy Kao Benefits of Partial Reconfiguration Take advantage of even more capabilities in your FPGA, Xcell Journal Xilinx, vol. I, pp. 65-67, 2005.

[20] Xilinx. MicroBlaze Processor Reference Guide : http: // www. xilinx. com/ support/ documentation/ sw_ manuals/ mb_ ref-guide. pdf , 2011.

[21] www.xilinx.com.

[22] Development System Reference Guide, v9.2i, Chapter 10, XPower.

[23] E. L. Horta , J. W. Lockwood , D. E. Taylor ,David Parlour, Dynamic Hardware Plugins in an FPGA with Partial Run Time Reconfiguration*, Design Automation Conference, DAC'02, 2002.*

[24] Two Flows for Partial Reconfiguration: Module Based or Difference Based, Xilinx Application Note XAPP290, version 1.1, Xilinx, Inc. 2003.

[25] M. Hübner, J. Becker, Exploiting dynamic and partial reconfiguration for FPGAs: toolflow, architecture and system integration*, Proceedings of the 19th annual symposium on Integrated circuits and systems design, tutorial session, 2006.*