Automatic synthesis of TTA processor networks from RVC-CAL dataflow programs

Jani Boutellier, Olli Silven, Mickaël Raulet

AUTOMATIC SYNTHESIS OF TTA PROCESSOR NETWORKS FROM RVC-CAL DATAFLOW PROGRAMS

J. Boutellier, O. Silvén

Computer Science and Engineering Laboratory
Department of Electrical and Information Engineering
FI-90014 University of Oulu

M. Raulet

INSA Rennes
IETR
FR-35708 Rennes Cedex 7

ABSTRACT

The RVC-CAL dataflow language has recently become standardized through its use as the official language of Reconfigurable Video Coding (RVC), a recent standard by MPEG. The tools developed for RVC-CAL have enabled the transformation of RVC-CAL dataflow programs into C language and VHDL (among others), enabling implementations for instruction processors and HDL synthesis.

This paper introduces new tools that enable synthesizing RVC-CAL dataflow programs automatically into application specific processor networks. Each processor in the network performs the functionality of one RVC-CAL actor. The processors are of the Transport Triggered Architecture (TTA) type, for which a complete codesign toolset exists. The existing tools enable customizing the processors according to the requirements of individual dataflow actors.

The functionality of the toolchain has been demonstrated by synthesizing an MPEG-4 Simple Profile video decoder to an FPGA. The decoder is realized into 21 processors that decode QCIF resolution video at 20 frames per second with a 50MHz FPGA clock frequency.

Index Terms— data flow computing, design automation, multiprocessor interconnection

1. INTRODUCTION

Signal processing systems constantly need more and more processing performance as their complexity increases. To tackle the increasing program complexity, algorithm designers have looked for new ways to express the programs in a less error-prone and more portable fashion than traditional imperative languages can offer. At the same time, processing hardware designers try to come up with processing platforms that offer better processing performance, scalability and energy-efficiency.

On the software side, a major step into this direction has been the CAL dataflow language [1]. A more restricted version of the CAL language, named RVC-CAL, has been standardized and used to specify the algorithms and functions required by the Reconfigurable Video Coding standard, which embodies the recent trends of software modularity, portability and concurrency [2].

Dataflow programs written in RVC-CAL can be compiled into implementation languages with the Open RVC-CAL Compiler (Orcc) that has several backends: C, C++, LLVM assembly and VHDL [3], which enable the algorithms specified in RVC-CAL to be executed on instruction processors or dedicated integrated circuits.

This paper presents a design flow that allows a designer of signal processing systems to write a program in the RVC-CAL dataflow language and automatically generate a multiprocessor implementation out of it. In addition to describing the algorithm in RVC-CAL, the designer is responsible for building the processors that are used in the system. The processor design is based on the TCE toolset, which has a graphical user interface and allows the designer to assemble processors without any HDL design skills. The design flow presented in this paper creates VHDL hardware descriptions as a final output and thus enables direct synthesis on FPGA boards. The functionality of the design flow is demonstrated with an MPEG-4 Simple Profile (SP) video decoder that is written in RVC-CAL.

2. BACKGROUND

Before going into the details of our design flow, the RVC-CAL language and the used processor technology are explained.

2.1. The RVC-CAL Language

RVC-CAL is a dataflow language. Dataflow languages are popular in signal processing system design, as they allow the designer to abstract the signal processing system to logical entities that interact with each other. It is up to the designer to decide how to partition the signal processing system into different dataflow entities, which are called *actors*.

A dataflow actor reads data from its inputs, performs some data processing and finally outputs the results. In RVC-CAL actors communicate with each other over FIFO buffers. The

set of actors interconnected with FIFOs is called an RVC-CAL *network*. The data is wrapped inside *tokens* and each FIFO carries tokens of a specified size. Internally, actors work like finite state machines (FSMs) that contain states, state transitions and internal variables.

Conditional execution is the most important feature of RVC-CAL when it is compared to traditional dataflow languages: for example, Synchronous Data Flow (SDF) [4] does not allow conditional execution.

2.2. Transport Triggered Architecture Processors

Transport Triggered Architecture processors resemble Very Long Instruction Word processors (VLIW) in the sense that they fetch and execute multiple instructions each clock cycle. A major difference, however, is that TTA processors have only one instruction: *move*, which simply transfers data from an input location to an output. For example, one move instruction can initiate a data transfer from the output of an *add* function unit (FU) to one of the inputs of a *mul* function unit.

In [5] it is stated that direct programming of the data transports reduces the register file traffic when compared to VLIWs, but on the other hand makes the compiler design quite challenging, as it is the compiler that schedules the data transports and makes sure that conflicts are avoided. As the compiler does so many decisions at design time, the run-time system is simplified and hence there are savings on the processor gate count and energy consumption.

The design of custom TTA processors has become easy and accessible to everyone through the open source TTA Codesign Environment (TCE) toolset [6, 7]. The TCE toolset offers a graphical user interface for custom processor, function unit and instruction design. The TCE toolset has a compiler which is based on LLVM[1] and contains a processor simulator and profiler. TCE also provides a possibility to realize the processors into VHDL files and memory images, which enable easy FPGA synthesis.

2.3. Related Work

There has been prior work similar to that presented in this paper. Park, Oh and Ha [8] list several multiprocessor system design methods that use various dataflow models as input. The difference of our work compared to the ones listed in the article is that our toolchain is the first one that supports the RVC-CAL language, and uses TTA processors.

In [9], a design flow is described for synthesizing heterogeneous multiprocessor systems out of CAL programs. The difference to our work is that the methodology in [9] does not target a specific platform, but remains on a more abstract level. In contrast, our work targets TTA processor networks and presents all necessary tools down to the level of FPGA synthesis.

[1] http://www.llvm.org/

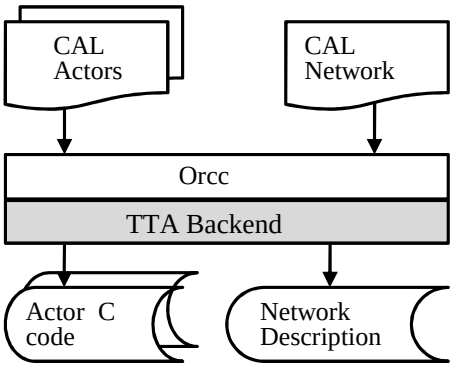

Fig. 1. Production of the (abstract) processor network description and the actors' C code.

3. PROPOSED SOLUTION

In this paper we propose a design flow that enables a signal processing system designer to write a program in the RVC-CAL language, and automatically produce an FPGA-ready multiprocessor system out of it.

The design flow requires three different inputs from the designer: 1) the actors, 2) the actor interconnection network, and 3) the processors. The actors are source code files written in the RVC-CAL language. An example of an actor is Inverse Discrete Cosine Transform (IDCT) that reads 8 tokens and produces 8 tokens of data. The actor interconnection network is an XML file that describes the connections between actors. Finally, the user needs to provide a processor specification for each RVC-CAL actor in the system. Thus, if the signal processing system consists of 15 RVC-CAL actors, the designer has to provide a total of 15 processors to the design flow. These processors can be identical or heterogeneous.

The actor files (written in RVC-CAL) and the actor network description file (written in XML) are processed by the Orcc compiler[2]. For our purpose, Orcc produces a C language file out of each RVC-CAL actor, as well as a network description file that is in a special format required by our design flow. This part of the design flow is depicted in Figure 1.

The processors are designed with the TCE (TTA Codesign Environment) toolset. The designer can create the processors with a graphical user interface without writing any hardware descriptions by hand. For each processor, the TCE toolset produces an Architecture Definition File (ADF) and a set of VHDL files. The ADF file is used by the TCE compiler to compile the C code of each actor into TTA machine language and produce memory images for the FPGA implementation. This part of the design flow is depicted in Figure 2.

To enable the design flow presented in this paper, some software tools and components had to be designed for Orcc and TCE. For inter-processor communication, special TTA

[2] http://orcc.sourceforge.net/

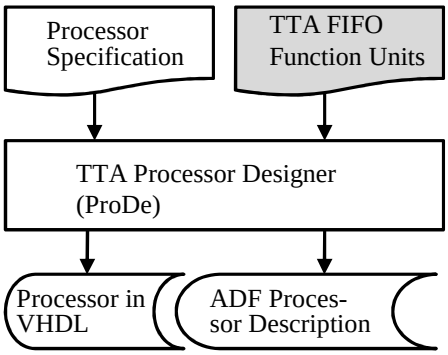

Fig. 2. The processor design in our design flow.

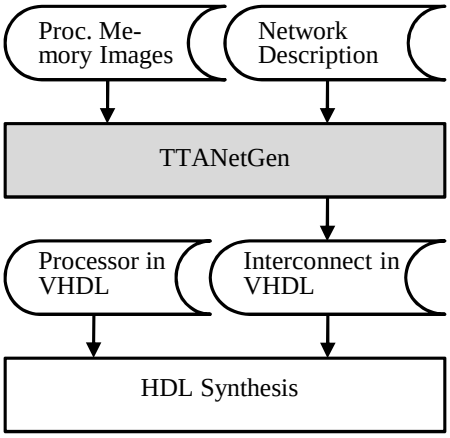

Fig. 3. Production of the interconnection network HDL description.

FIFO functional units had to be designed to enable the processors to communicate over hardware FIFOs (see Figure 2). For Orcc, a special TTA backend had to written (see Figure 1). The TTA backend produces C code that is specially meant for TTAs that contain FIFO function units. The network description produced by the TTA backend is processed by the *TTANetGen* tool (Figure 3) that generates the interconnect between the TTA processors. Next, these components and tools are described in detail.

3.1. The TTA FIFO Function Units

The special function units (FUs) for accessing external FIFO memories were designed to enable inter-processor communication with minimal overhead. The "FIFO read" TTA FU implements three instructions: *status* (returns number of tokens in the FIFO), *read* (reads a token) and *peek* (shows the value of the next token in the FIFO). The "FIFO write" TTA function unit, on the other hand, has just *status* and *write* instructions.

The instructions were implemented in C++ for the processor simulator *ttasim* and in VHDL for FPGA implementations. The latencies of these new functions units range between 1 and 3 clock cycles depending on the instruction.

3.2. The TTA Backend for Orcc

The backends of the Orcc compiler are easy to customize, as they are specified with StringTemplate (http://www.stringtemplate.org/) files that can be modified with any text editor. To make each actor directly multiprocessing-capable, the Orcc C language backend was modified such that it produces actors that access FIFOs with their special instructions that were introduced in Subsection 3.1.

Each FIFO connection of an actor is given an index number which directly invokes a TTA function unit that is connected to the respective hardware FIFO. Thus, if a processor executes actor A, and actor A has 5 input ports, the processor has 5 separate function units that are directly connected to 5 different FIFOs.

It is also worth mentioning that using C code as an intermediate language between Orcc and TCE is not mandatory. Actually Orcc is capable of directly producing LLVM assembly [10], which is already an intermediate language of TCE. Using LLVM assembly as a bridge between Orcc and LLVM is a natural direction of future work.

3.3. Generating the Interconnect Between Processors

In general, the complexity of actors in an RVC-CAL network can vary considerably from actor to actor. As each RVC-CAL actor is running on a separate TTA processor, it is not sensible to reserve the same amount of program memory, data memory or computational resources (such as multipliers) for each processor.

The TTA Processor Designer (ProDe) software produces instruction and data memory images for each processor. The sizes of these memory images are analyzed by our *TTANetGen* software, which produces VHDL envelopes for each memory unit and processor, so that the processors with their private memories are encapsulated into single entities at the top level design file.

Finally, *TTANetGen* generates the top-level VHDL file for the whole system. In the first phase, the output of Orcc is analyzed to generate an internal representation of the actor network. In the second stage this internal representation is written to a top-level VHDL file that has each FIFO and each processor as a separate entity. This part of the design flow is depicted in Figure 3.

4. DESIGN FLOW

In this section we describe the use of the design flow that is described in this paper.

We assume that in the very beginning the user has only an abstract idea of the program that she or he wishes to imple-

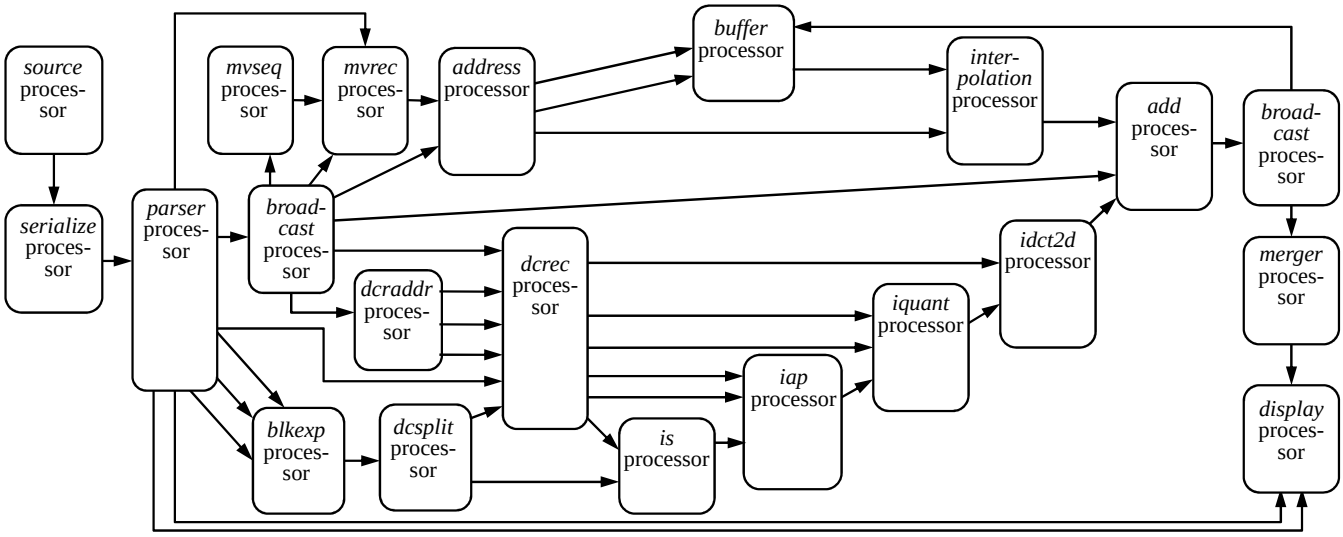

Fig. 4. The TTA processor network that has been synthesized from the MPEG-4 SP RVC-CAL dataflow program. The arrows between the processors are FIFO buffers.

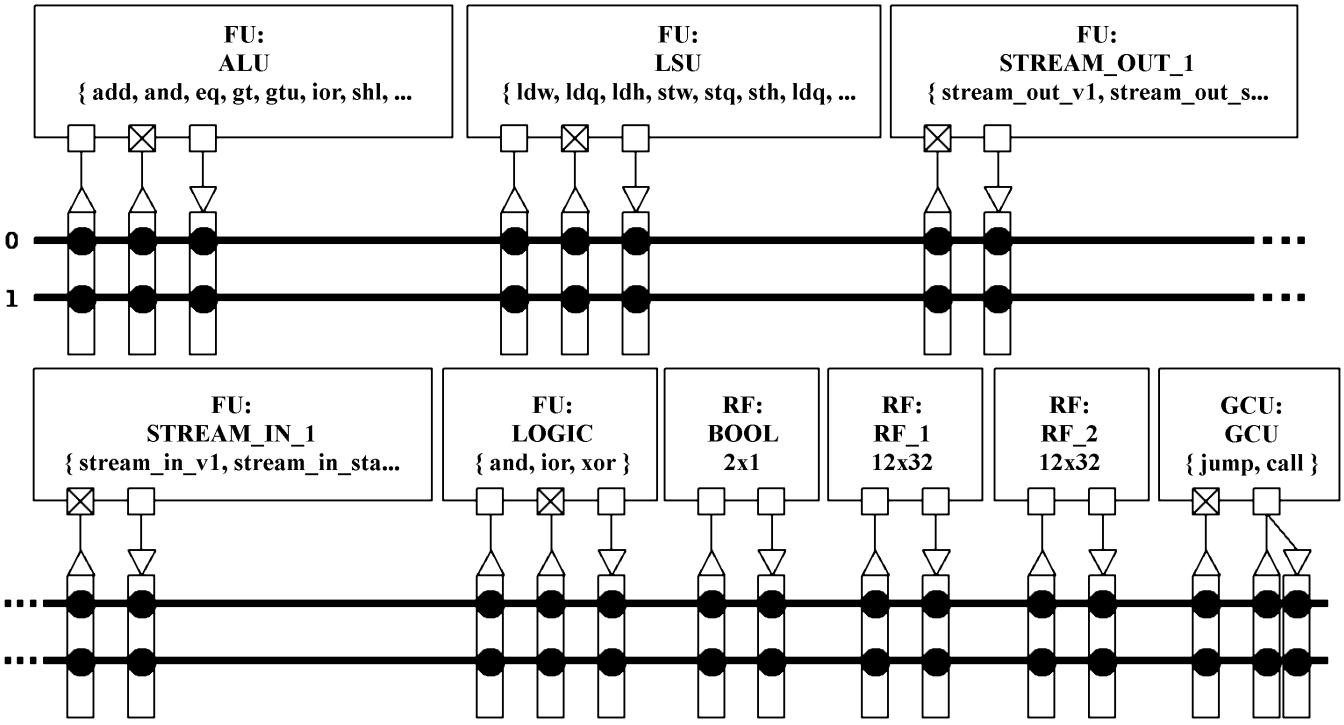

Fig. 5. The TTA processor that executes the *serialize* actor. The processor has 2 transport buses, 2 general register files (RF) with sizes of 12 slots each. STREAM_IN and STREAM_OUT are the special FIFO function units.

ment with the design flow. The very first task the user needs to do, is to split his program into logical entities. The division of the application into entities is a very important part, because it will later on affect the maximum attainable performance, as well as the resource consumption on the target device.

After the partitioning of the abstract program the user needs to write the functionality of the individual parts as RVC-CAL actors, and draw data connections between the actors where ever they need to communicate. The Orcc compiler exists as a plugin to the Eclipse[3] development environment and offers a text editor that assists in writing of RVC-CAL. Finally, the user can use the Orcc compiler to produce the C code implementations and interconnection information of the actor network. This phase was depicted in Figure 1.

As it is quite hard to predict the computational resource requirements of individual actors, it is recommended that the user designs one low-performance processor that he uses as the initial execution entity for each actor. This processor must be designed such that it fills the minimum needs of every actor, but does not offer superfluous resources.

Next, the user provides the C code implementations of the actors together with the processor description (ADF) to the TCE compiler, which produces binary files for each actor. Having the actor binaries, the user can now profile the performance of actors on the initial processor. If necessary, the user can create more powerful custom processors for actors that need more performance. One of the most important ways to improve TTA processor performance is adding more transport buses. Each transport bus in the processor can perform one data move each clock cycle.

As shown in Figure 2, the user produces the VHDL and ADF description of the processor. As the processor network generally forms a sort of a pipeline, it is desirable that all processors have an identical latency in the final implementation. Profiling can easily be performed with the cycle-accurate TTA simulator *proxim* that belongs to the TCE toolset.

After the user has customized the processors and reached the performance requirements, she or he can use the TCE processor generator to produce VHDL implementations of each processor. The VHDL description of the interconnect between processors, on the other hand, is produced by our *TTANetGen*. By default, the design flow offers a special *source* actor that can be used to input processing data to the network from a dedicated on-chip memory. If the user wishes to use another kind of a data source, he must at this stage design it. Likewise, the default data output is the *display*, which enables directing the computation results to FPGA general output pins (such as LEDs). For the actual synthesis, the user can use his favourite design environment. However, the interfaces to on-chip memories (required by FIFOs, and processor memories) have been designed only Altera devices in mind.

[3]www.eclipse.org

Table 1. Instruction memory and data memory requirements of actors.

Entity	*I. Words*	*Buses*	*DMem Depth*
Add	334	2	91
Address	844	6	118
Broadcast VID	29	2	87
Buffer	107	2	22265
Interpolation	500	2	88
Block Exp.	188	2	94
MV Rec.	1217	2	176
MV Seq.	569	2	97
Parser	3707	2	980
Broadcast BTYPE	53	2	87
Serialize	53	2	89
DC Rec. Addr.	597	2	209
DC Rec. I. Pred.	859	2	157
DC Split	49	2	88
Inv. AC Pred.	249	6	992
IDCT 2D	1085	6	153
Inv. Quant.	154	2	92
Inv. Scan	578	2	203
Display	232	2	94
Source	13	2	87
Merger	930	2	1117

5. EXPERIMENTS

To demonstrate the functionality of our design flow, we took the RVC-CAL program describing an MPEG-4 Simple Profile decoder and synthesized it using our design flow as a multi-TTA processor network. The total number of generated processors was 21.

The synthesis results were transferred to the Altera Cyclone IV EP4CE115F29C7 FPGA. We used a video stream of QCIF resolution as test data to verify the correct operation of the network. The decoding result was compared against a checksum that had been computed for each frame on a workstation beforehand.

All of the memories and video bitstream fit on the FPGA on-chip resources (EP4CE115 has 432 kB of on-chip RAM), when the maximum video resolution was limited to QCIF. Higher video resolutions would make the size of the prediction buffer explode and would require resorting to off-chip memory, which was not done at this stage.

Table 1 shows the resources consumed by each processor for the MPEG-4 SP decoder. The most notable figures are the instruction memory usage of the *parser* and the data memory usage of *buffer*. Parser is by far the most complex actor in the network and it requires almost 4000 instruction words in a two-bus TTA processor. The buffer actor, on the other hand, is very simple, but its large data memory contains the prediction buffer.

Table 2. Total resource usage.

Resource	*Used*	*Available*	*%*
Logic Elements	66'762	114'480	58%
1kB Mem. Blocks	427	432	99%
Block Interconnects	108'511	342'891	32%

Table 3. Performance

Unit	*Clock Cycles*
Intra frame	2'904'000
Pred. frame	29'300
Intra macrob.	2'633'000
Pred. macrob.	26'600

Most of the processor instances in the network are identical. As the tools that form the basis of the toolchain, TCE and Orcc, allow creating heterogeneous processors, this was done for some actors that required more than an average amount of resources. The processors that required more performance can be seen in Table 1: they have more than 2 transport buses. On the contrary, Figure 5 shows a tiny TTA processor that was used to execute the code of the *serialize* actor.

Table 3 shows the number of clock cycles consumed by intra frames and predicted frames, which was well-balanced due to the possibility to tune the performance of each processor. It was detected that the main limiting factor in the system performance was in the backend that generates code for TTAs. The TTA processors could compute much faster, but the automatically generated code still had some unnecessary computations at this stage. Improving this is a clear direction for future work.

Regarding the FPGA resource usage (see Table 2), the example application fit well to the FPGA board. The TTA feature of *instruction word compression* helped alleviate need of scarce on-chip memory resources. Through instruction word compression, it was possible to save on the usage of memory blocks with the expense of logic element usage.

6. CONCLUSION

In this paper we have presented a design flow that enables automatic synthesis of RVC-CAL actor networks to application specific transport-triggered processor networks. The functionality of the toolchain has been demonstrated by applying it to an RVC-CAL network, which defines an MPEG-4 SP video decoder. The performance of the resulting implementation was 20 frames per second on a 50 MHz FPGA and QCIF video resolution.

7. ACKNOWLEDGEMENTS

The authors would like to thank Pekka Jääskeläinen and Otto Esko for technical help related to TTAs.

8. REFERENCES

[1] J. Eker and J. Janneck, "CAL language report," Tech. Rep. UCB/ERL M03/48, UC Berkeley, 2003.

[2] M. Mattavelli, I. Amer, and M. Raulet, "The Reconfigurable Video Coding standard," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 159–167, May 2010.

[3] Nicolas Siret, Matthieu Wipliez, Jean-François Nezan, and Aimad Rhatay, "Hardware code generation from dataflow programs," in *Proceedings of the 2010 Conference on Design & Architectures for Signal & Image Processing*, 2010, pp. 113–120.

[4] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept. 1987.

[5] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*, John Wiley & Sons, Inc., New York, NY, USA, 1997.

[6] P. Jääskeläinen, V. Guzma, A. Cilio, and J. Takala, "Codesign toolset for application-specific instruction-set processors," in *Proc. SPIE Multimedia on Mobile Devices*, San Jose, CA, January 2007, pp. 65070X–1 – 65070X–11.

[7] O. Esko, P. Jääskeläinen, P. Huerta, C.S. de La Lama, J. Takala, and J.I. Martinez, "Customized exposed datapath soft-core design flow with compiler support," in *International Conference on Field Programmable Logic and Applications*, 2010, pp. 217–222.

[8] Hae woo Park, Hyunok Oh, and Soonhoi Ha, "Multiprocessor SoC design methods and tools," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 72 –79, november 2009.

[9] C. Lucarz, G. Roquier, and M. Mattavelli, "High level design space exploration of RVC codec specifications for multi-core heterogeneous platforms," in *Conference on Design and Architectures for Signal and Image Processing*, oct. 2010, pp. 191–198.

[10] J. Gorin, M. Wipliez, F. Prêteux, and M. Raulet, "LLVM-based and scalable MPEG-RVC decoder," *Journal of Real-Time Image Processing*, vol. 6, pp. 59–70, 2011, 10.1007/s11554-010-0169-2.