Fast Hardware implementation of an Hadamard Transform Using RVC-CAL Dataflow Programming

Khaled Jerbi*†, Matthieu Wipliez*, Mickaël Raulet*, Olivier Déforges*, Marie Babel* and Mohamed Abid†
*IETR/INSA. UMR CNRS 6164, F-35043 Rennes, France, mail: Firstname.Name@insa-rennes.fr
†CES Lab. National Engineering school of Sfax, Tunisia, mail: Firstname.Name@enis.rnu.tn

***Abstract*—Implementing an algorithm to hardware platforms is generally not an easy task. The algorithm, typically described in a high-level specification language, must be translated to a low-level HDL language. The difference between models of computation (sequential versus fine-grained parallel) limits the efficiency of automatic translation. On the other hand, manual implementation is time-consuming, because the designer must take care of low-level details, and write test benches to test the implementation's behavior. This paper presents a global design method going from high level description to implementation. The first step consists in describing an algorithm as a dataflow program with the RVC-CAL language. Next step is the functional verification of this description using a software framework. The final step consists in an automatic generation of an efficient hardware implementation from the dataflow program. The objective was to spend the most part of the conception time in an open source software platform. We used this method to quickly prototype and generate hardware implementation of the Hadamard transform, an algorithm used in many signal processing algorithms, from an RVC-CAL description.**

I. Introduction

Signal processing algorithms are increasing in complexity. This complexity involves a very long description code. For designers this code is very hard to implement on hardware platforms. Hardware implementation requires the description of the process using an HDL language like VHDL or Verilog. These dataflow languages are not easy to develop and especially to validate. The validation of a dataflow design requires the development of stimulus code such as a VHDL test bench in our case and the use of simulation tools. This is what explains the elapsing gap between validating and implementing a process. Therefore designers can hardly satisfy the time to market constraints. To solve this problem, designers are establishing solutions to describe the process in a higher level way. In the video coding field, a new high level description language for dataflow applications called RVC-CAL [1] was normalized by the MPEG community through the MPEG-RVC standard [2]. This standard provides a framework to define different codecs by combining communicating blocks developed in RVC-CAL.

The objective of our work is the hardware generation from an RVC-CAL description [3]. In this paper, we introduce an original global approach to fasten the validation of an RVC-CAL design and consequently the dataflow generation. This approach was applied on the Hadamard transform used in several signal processing algorithms. In section II, we present the approach and the used languages and frameworks. Section III shows an application of the method on the Hadamard transform and also provides some implementation results.

II. Dataflow programming for hardware implementation

The purpose of this work is to obtain a dataflow description directly from an RVC-CAL design. Presently, the only hardware generator from CAL is a tool called Cal2HDL [4], [5]. It uses an intermediate representation of the OpenDF project [6]. Nevertheless,this tool is still unable to treat with all the RVC-CAL structures (loops and repeats). Therefore, the existing development method consists of developing a CAL code synthesizable with Cal2HDL. Then this code is validated through the OpenDF simulator and finally synthesized into Verilog/VHDL using Cal2HDL. The limitation is the fact that a synthesizable code is very long and accordingly so difficult to manage and to correct. In addition, the feedback of the OpenDF simulator and the HDL generator are not accurate enough. So the errors correction is therefore relatively a long task.

In the following, we present a new approach for functional verification of an RVC-CAL code. As presented in figure 1, the design is described with a high level RVC-CAL. Then a software platform is used for functional validation and FIFO sizing. Once the code is correct, it undergoes a modification to be compatible with Cal2HDL by unrolling the loops and the repeat structures. The validation of this code is realized with the same software platform. Before implementing the design, Cal2HDL provides an important feedback about the delay of every action in every actor. The implementation is finally insured using a hardware synthesis and prototyping platform.

A. Dataflow programming with RVC-CAL language

MPEG RVC is under development as part of the MPEG-B standard [3], which defines the framework and the language used to describe components. RVC-CAL [3] is a textual and domain specific language for writing dataflow models (figure 2), more precisely for defining *actors* of a dataflow model at a high level description. An actor represents an autonomous entity and a composition of actors explicitly describes the concurrency of an application. The RVC-CAL Actor Language has been defined to be platform independent and retargetable to a rich variety of platforms.

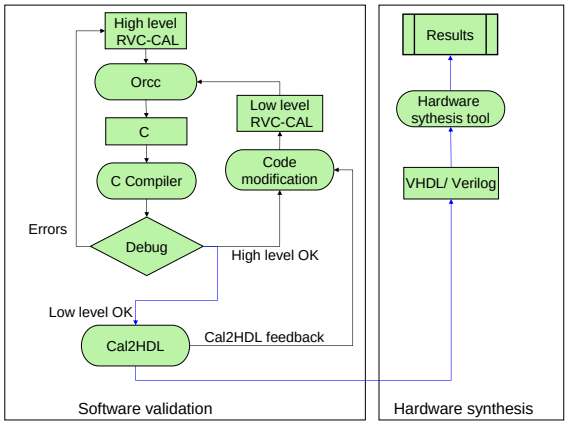

Fig. 1. Method overview

An RVC-CAL *actor* is a computational entity with input ports, output ports, states and parameters. An *actor* communicates with other actors by sending and receiving *tokens* (atomic pieces of data) through its ports. An actor can contain several *actions*. An *action* defines a computation, which consumes sequences of tokens from input ports and produces sequences of tokens to output ports. Actions have data-dependent conditions for their execution. The execution of an action may change the actor internal state, so that the produced output sequences are functions of the consumed input sequences and of the current actor state. RVC-CAL supports higher-level constructs such as multiple-token reads/writes, and list generators.

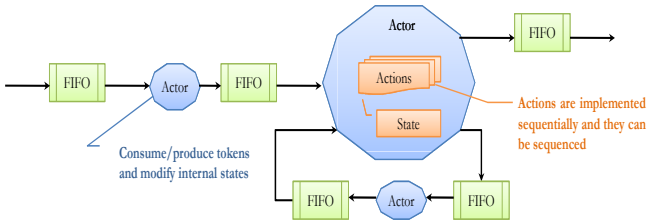

Fig. 2. CAL dataflow model

B. Functional verification on a software platform

CAL code validation is usually based on the OpenDF simulator. It has to be stimulated with manually given tokens via data generation and data display actors. The result is a set of values that have to be verified. The originality of our approach is to realize the CAL validation step using Open RVC-CAL Compiler (Orcc) [7]. orcc Compiler is an opensource software (http://sourceforge.net/projects/Orcc/) developed in the IETR laboratory of the INSA of Rennes. The orcc Compiler is a source-to-source compiler that compiles RVC-CAL dataflow programs to a target language. Available languages include C, C++ and Java. This compilation is obtained through intermediate transformations. First the CAL code is parsed for syntactic and semantic analysis. Then this analysis leads to an intermediate representation. Finally the analysis of the representation results in the target language. We use the C language in our work. After compilation, we can easily assign a video or an image as an input and visualize the output.

It is very important to mention that Orcc compilation, video processing and display using the C compiler are very fast steps. In addition, the software debug is very fast and efficient. Consequently, the CAL errors are easier detected and corrected. Moreover, we can use Orcc to define the optimal FIFO sizes for a lower memory consumption in the hardware implementation.

C. HDL generation

Dataflow generation is done with a tool called Cal2HDL. This tool parses the CAL code, generates an XML representation for each actor and synthesizes the static single assignment (SSA) threads into circuits based on basic operators. The final description is made up of a verilog file for each actor and a VHDL file for the top. The connection between the actors is insured by synchronous or asynchronous FIFO buffers.

For the moment, Cal2HDL does not support all the structures used in RVC-CAL description such as repeats and loops. These structures have to be manually modified into several actions managed by finite state machine. Figure 3 shows an exeample of an action writing the 16 values of a buffer named ”tab” in the output port called ”OUT”. The instruction ”repeat 16” enables the access to the 16 first values of the buffer ”tab”.

```
write: action ==> OUT:[tab] repeat 16
end
```

Fig. 3. High level RVC-CAL example

This action has to be modified into the code presented in figure 4. The modifications consist of deleting the ”repeat” structure to have an action that produces only one token and repeats the basic action 16 times. The repetition process starts by executing the ”write” action until the ”write_done” action is validated. Everything has to be managed by a finite state machine defined by the structure ”schedule fsm” in figure 4.

After this transformation we obtain a synthesizable code and Cal2HDL can generate the adequate hardware description.

III. Hardware implementation of an Hadamard transform

The Hadamard is a transform used in many image and video coders notably the LAR (Locally Adaptive Resolution) [8], a coder developed in the IETR/INSA laboratory. It is composed of two layers: the spatial coder called the Flat LAR and the texture coder based on the Hadamard transform. The Flat LAR has already been developed with RVC-CAL and implemented in a previous work [9]. Therefore, from this preliminary implementation we would almost achieve the

```
write: action ==> OUT:[out]
do
  counter := counter + 1;
end

write_done: action ==>
guard
  counter = 16
do
        counter := 0;
end

schedule fsm write:
  write (write     ) --> write;
  write (write_done) --> nextstate;
...
end
```

Fig. 4. low level RVC-CAL example

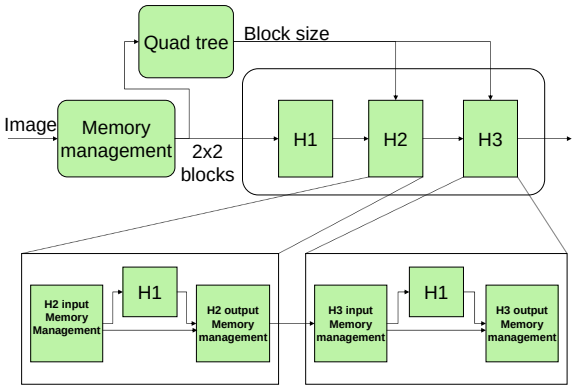

Fig. 5. Hadamard developed model

implementation of the whole LAR codec following the MPEG-RVC standard recommandations.

This section explains the mechanism of the Hadamard transform, the dataflow implementation and the synthesis results obtained when considering the methodology described above.

A. Hadamard transform

The Hadamard transform derives from a generalized class of the Fourier transform. It consists of a multiplication of a 2^mx2^m matrix by an Hadamard matrix (H_m) that has the same size. Here are examples of Hadarmard matrices:

$$H_0 = 1 \ , \ H_1 = \frac{1}{\sqrt{2}} \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} \ , \text{ etc ...}$$

Hadamard matrices are reversible. Therefore, the Hadamard transform is reversible and consequently the coding can be losslessly realized.

B. Hardware implementation

The LAR coding is dependent from the content of the image. It applies in the Flat LAR a morphological gradient to extract information about the local activity on the image. Then follows a Quadtree structure whose resulting output is the block size image represented by variable size blocks (2x2, 4x4 or 8x8). The higher the activity, the lower the block size is. We integrated the Quadtree structure in the Hadamard design because this image is used farther to apply the adequate transform on the corresponding block. It means that if we have a block size of 2X2 in the size image this block will undergo a 2X2 Hadamard (H_1) and a normalization specific to the 2X2 blocks.

The implemented Hadamard transform is presented in figure 5.

As a first step, the memory management block stores the pixels values of the original image line by line. Once an 8x8 block is obtained, the actor divides it in sixteen 2x2 blocks and sends them in a specific order as presented in figure 6. This order is very important to improve the performance of remaining actors. Then, in the Quadtree, a morphological gradient is applied to pull out the local activity and generate the block size image.

We also notice that an (H_2) transform can be achieved using the (H_1) results of the four 2X2 blocks constituting the 4X4 block. Idem for the (H_3) one. This ascertainment is very important for decreasing the complexity of the process. In fact, the Hadamard transform of the LAR applies an (H_1) transform for the whole image then it applies the (H_2) transform only for the 4X4 and 8X8 blocks and the (H_3) transform only for the 8X8 blocks. The (H_2) and the (H_3) transforms are different from the full transforms as they are much less complex. Consequently, as shown in figure 5, we designed the H2 and the H3 using H1 actors associated with memory management units. They sort tokens in the adequate order and, considering the block size, whether the block is going to undergo the transform or not.

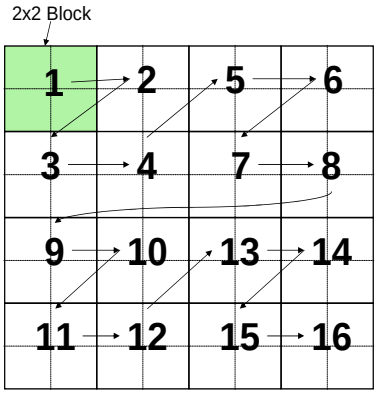

Fig. 6. Memory management unit output order

The different functions of the Hadamard have been first developped with a high level RVC-CAL description for a 352x288 image size. To optimize the transform, we added a ping-pong data management algorithm. The principle of this algorithm is to avoid the latency caused by data storage by combining the tokens reading and writing in the same action.

The tip is to write the input data in the half of a memory size then to use this data while writing in the other half. Finally we just have to switch the reading and the writing pointers in opposite from a half memory to the other.

A reverse Hadamard block was added for validation. The whole design was compiled with Orcc to obtain the C code of the actors. C code were compiled with a C compiler. To test the design we applied images and videos in the inputs. The objective was to obtain an output exactly equal to the output as presented in figure 7.

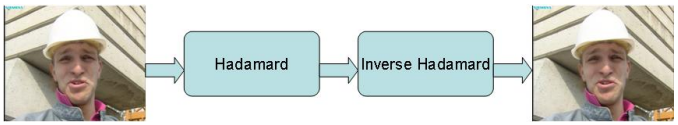

Fig. 7. Software validation

Once the required pixel values are obtained the design is validated and consequently the RVC-CAL code. At this level, the VHDL/Verilog generation is not possible since Cal2HDL can not generate code from the high level RVC-CAL. It was necessary to change the RVC-CAL code into another low level code synthesizable with Cal2HDL as explained in Section II. Thus, we obtained a dataflow implementation of the Hadamard transform.

C. Results

The HDL project manager environment used is Xilinx ISE Foundation 11.1 and the hardware simulation tool is ISE simulator (Full version). We develop the testbench manually by initializing the different signals and generating the stimulus values for the inputs.

After compilation, simulation, RTL synthesis and place and route on an FPGA: family=virtex4; device=xc4vsx35; Package=FF668; speed = -12, we obtain the area consumption results presented in table I.

Criterion	value
Slice Flip Flops	1,244/30.720 (4%)
Occupied Slices	2,019/15.360 (13%)
4 input LUTs	3,464/30.720 (11%)
FIFO16/RAMB16s	29/3192 (15%)
Bonded IOBs	47/448 (10%)

TABLE I
CONSUMPTION FOR 352X288 IMAGE SIZE

The time synthesis performances are mentioned in table II Optimization solutions are in development to decrease the latency and consequently increase the frequency. In terms of development time, the whole design took about 60 days to be achieved. It is very important to mention that over 90% of the conception time was achieved in the open source software platform. The most disturbing part of the flow was the manual transformation of the RVC-CAL from high to low level. We are currently looking for solutions to automate this step. However, this global framework introducing a software functional checking before the synthesis process is significantly faster than a hardware implementation directly from the RVC-CAL description.

Criterion	352x288
Output frequency(MHz)	22.4
maximum frequency(MHz)	112
Latency(μs)	79.4
processing time(ms)	4.5
Minimum input arrival time before clock(ns)	4.339
Maximum output required time after clock(ns)	8.001
Maximum combinational path delay(ns)	5.083

TABLE II
TIMING RESULTS

IV. CONCLUSION

This paper presented a method to automatically generate an efficient functional hardware implementation from an RVC-CAL dataflow program. The presented method was used to obtain a hardware implementation of a variable-size Hadamard transform. We believe that frequency can be increased, and latency decreased, by further optimization of memory management actors.

With our method, the design cycle of a hardware implementation consists in doing the functional verification in software, and testing the hardware implementation once the program is correct. We used the Orcc Compiler to generate C code from RVC-CAL descriptions and to fix the optimal FIFO sizes. The C code was then compiled and run to test the program behavior. The hardware implementation was obtained by automatically transforming the RVC-CAL descriptions with Cal2HDL. Currently, high-level RVC-CAL descriptions must be manually transformed to lower-level code for Cal2HDL to be able to synthesize it. Automating this transformation will further reduce design time and will be a direction of future works.

REFERENCES

[1] J. Eker and J. Janneck, "CAL Language Report," University of California at Berkeley, Tech. Rep. ERL Technical Memo UCB/ERL M03/48, Dec. 2003.

[2] 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, dOI:10.1007/s11265-009-0399-3. To appear.

[3] ISO/IEC FDIS 23001-4: 2009, "Information Technology - MPEG systems technologies - Part 4: Codec Configuration Representation," 2009.

[4] 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," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 11, pp. 1646–1657, 11 2009. [Online]. Available: dx.doi.org/10.1109/{TCSVT}.2009.2031517http://hal.archives-ouvertes.fr/hal-00440492/en/

[5] "Cal2hdl-openforge source : http://openforge.sourceforge.net."

[6] 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," 2008, first Swedish Workshop on Multi-Core Computing, MCC , Ronneby, Sweden, November 27-28, 2008.

[7] 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, NY, USA: ACM, 2010, pp. 223–234.

[8] O. Déforges, M. Babel, L. Bédat, 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.

[9] K. Jerbi, M. Raulet, O. Déforges, and M. Abid, "Design of an Embedded Low Complexity Image Coder using CAL language," *DASIP 2009 proceeding*, September 2009.