Automatic Method For Efficient Hardware Implementation From RVC-CAL Dataflow: A LAR Coder baseline Case Study

Khaled Jerbi, Matthieu Wipliez, Mickaël Raulet, Olivier Déforges, Marie Babel, Mohamed Abid

Automatic Method For Efficient Hardware Implementation From RVC-CAL Dataflow: A LAR Coder baseline Case Study

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 of the description of 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 of 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 a baseline part of the LAR coder, 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 a hardware implementation generated from a high level description using RVC-CAL programing language. [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 LAR (Locally Adaptive Resolution) image coder [4]. The actual design does not contain the full LAR coder, but we already achieved some main parts with an RVC-CAL description. This is why, in the following, we are going to speak about a LAR coder baseline.

In section II, we present the approach and the used languages and frameworks. In section III, the LAR coding principle is detailed. Section IV shows an application of the method on the LAR coder baseline and also provides some implementation results. Finally in section V some related works will be presented.

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 [5], [6]. It uses an intermediate representation of the OpenDF project [7]. Nevertheless, this tool is still unable to treat with all the RVC-CAL structures. It cannot handle with loops and repeats. Therefore, the existing development method consists of developing an RVC-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. They just mention an error without localizing it in the code's lines. So the errors correction is therefore relatively a hard task if the code is long.

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 synthesizable 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.

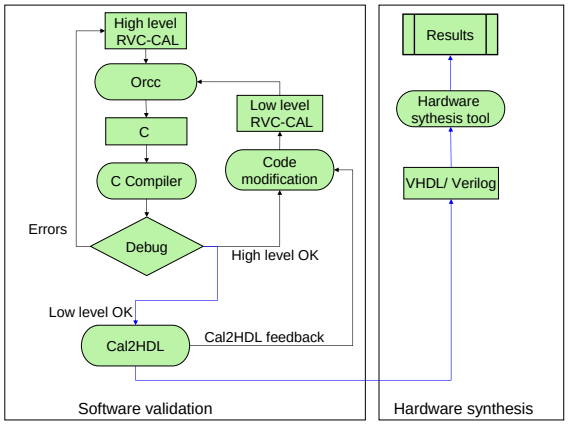

Fig. 1. Method overview

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.

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.

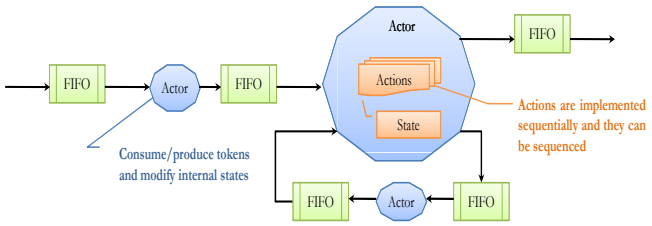

Fig. 2. CAL dataflow model

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) [8]. Orcc Compiler is an opensource software (http://sourceforge.net/projects/Orcc/) developed at 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. In our work, we use the C backend of Orcc. This choice is explained by the fact that C language is the most used language in software programming. 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 faster corrected. Moreover, we can use Orcc to define the optimal FIFO sizes for a lower memory consumption in the hardware implementation. To adjust FIFO sizes, we have to start by computing the minimum size by considering the data rate sent by the previous actor. Then this size is incremented until a correct video display is obtained.

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.

Currently, 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.

```
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

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

III. The LAR coder

The LAR coder is developed at the IETR/ INSA of Rennes laboratory. It is based on the idea that the spatial coding can be locally dependent on the activity in the image. Thus, the higher the activity the lower the resolution is. This activity is dependant from the variation or the uniformity of the local luminance which can be detected using a morphological gradient that will be farther explained. Another aspect of the LAR coding is based on considering that an image is a superposition of a global information image (mean blocks image), and the local texture image, which is given by the difference between the original image and the global one. This principle is explained by:

I = I’ + (I-I’) where I is the original image, I’ is the global information image and (I-I’) is the error image. The dynamic range of the error image is consequently dependent on the local activity. In uniform regions, I’ values are close or equal to I consequently (I-I’) values are around zero with a low dynamic range.

Considering these principles, the LAR coder concept (figure 5) is composed of two parts: the FLAT LAR [9] which is the part insuring the global information coding and the spectral part which is the error spectral coder.

The mecanisms of these parts are detailed in the following.

A. FLAT LAR

The Flat LAR is composed of 3 main parts: the partitioning, the block mean value and the DPCM (Differential Pulse Coding Modulation). In our work, only the DPCM is not yet developped with RVC-CAL.

1) Partitioning: In this part, a Quad-Tree partitioning is applied on the image pixels. The principle is to consider the lowest block size (2x2) then to compare the difference between the maximum (MAX) and the minimum (MIN) values of the

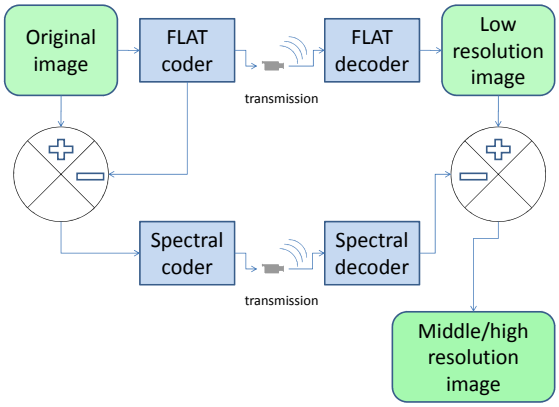

Fig. 5. LAR concept

block with a threshold (THD) defined as a generic variable for the design. If (MAX - MIN) < THD then the actual block size is considered. In the other case, the (Nx2) x (Nx2) size block is adapted. This process is recursively applied on the whole image blocks. The output is the block size image.

2) Block mean values: This process is based on the Quad-Tree output image. For each block of the variable size image, a mean value is put in the block as presented in the example of figure 6.

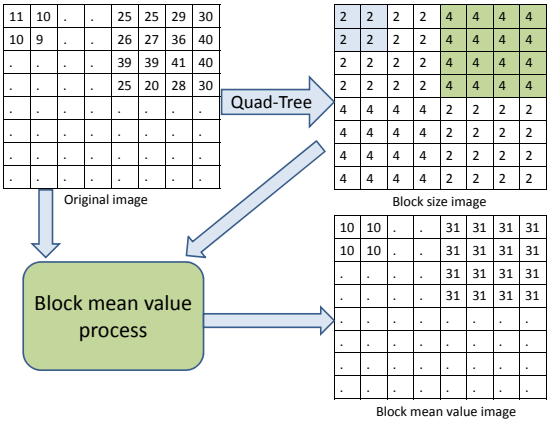

Fig. 6. Block mean value process example

3) The DPCM: The DPCM process is based on the prediction of neighbor values and the quantization of the block mean value image. The observation that a pixel value is mostly equal to a neighbor one led to the following estimation algorithm. If we consider the pixels in figure 7, X value is estimated with the algorithm: If |B-C| < |A-B| then X = A else X = C

B. Spectral coder: The Hadamard transform

The spectral coder, also called the texture coder, is composed of a variable block size Hadamard transform [10] and

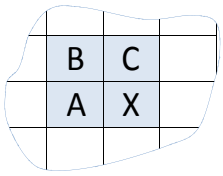

Fig. 7. DPCM prediction of neighbor pixels

the Golomb-Rice [11] [12] enthropic coder. The Golomb-Rice coder is still in development with the RVC-CAL specifications.

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. The transform is defined as:

H_0 is the identity so $H_0 = 1$. For any m>0, H_m is then deducted recursively by:

$$H_m = \frac{1}{\sqrt{2}} \begin{vmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{vmatrix}$$

Here are examples of Hadarmard matrices:

$$H_0 = 1 \ ,$$

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

$$H_2 = \frac{1}{\sqrt{2}} \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{vmatrix} \ , \text{ etc ...}$$

IV. Hardware implementation of the LAR coder baseline

Some parts of the Flat LAR have already been developed with RVC-CAL and implemented in a previous work [13]. Therefore, from this preliminary implementation we would almost achieve the implementation of the whole LAR codec following the MPEG-RVC standard recommandations.

This section explains the mechanisms of the Hadamard transform and the Quad-Tree used in the implementation. Dataflow implementation and synthesis results are presented and discussed.

A. Hardware implementation

The LAR coding is dependent from the content of the image. It applies in the Quad-Tree a morphological gradient to extract information about the local activity on the image. The 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. Using the block size image, the Hadamard transform applies 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, idem for 4X4 and 8X8. A size specific contization step is applied on the Hadamard output image. For each block size, a quantization matrix is predefine. Practically, the normalization during the Hadamard transform is postponed to be achieved with the quantization step to decrease the noise due to successive divisions.

The implemented LAR is presented in figure 8.

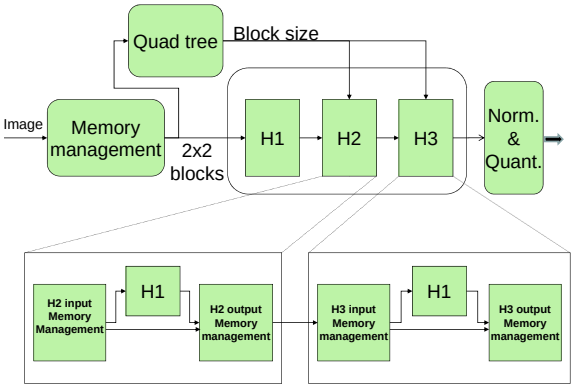

Fig. 8. LAR baseline developed model

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 into sixteen 2x2 blocks and sends them in a specific order as presented in figure 10.

This order is very important to improve the performance of remaining actors. In fact, considering the figure 10, when the tokens are so ordered the first 4 tokens are the first 2x2 block, the first 16 tokens are the first 4x4 block etc ... Consequently, and as presented in figure 8, the output of the H1 is automatically the input of the H2 and the output of the H2 is automatically the input of the H3.

In the Quad-Tree, this order is also crucial. As presented in figure 9, the superposition of the same actor (max for example) three times provides in the output of the first actor the maximums of 2x2 blocks, in the output of the second actor the maximums of 4x4 block and finally the maximums of 8x8 blocks in the output of the third one. Using the maximums and the minimums the morphological gradient in the Gradstep actors can process to extract the block size image. The same tip is used to calculate the block sums with three superposed sum actors. The block mean value actor considers the sums and the sizes to build the block mean value 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

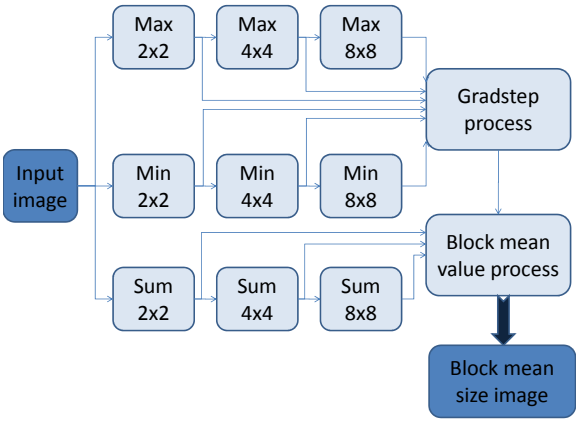

Fig. 9. Quad-Tree design

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 8, 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.

It is very important to mention that almost actors have been developed with generic variables for memory sizes or gradsteps which means that the design are flexible for easy transformation from an image size to another or for adding higher Hadamard process (H4, H5 ...).

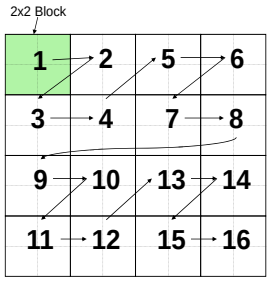

Fig. 10. Memory management unit output order

The different actors of the LAR baseline 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. An example of ping-pong memory management of a 4 buffers memory is presented in figure 11.

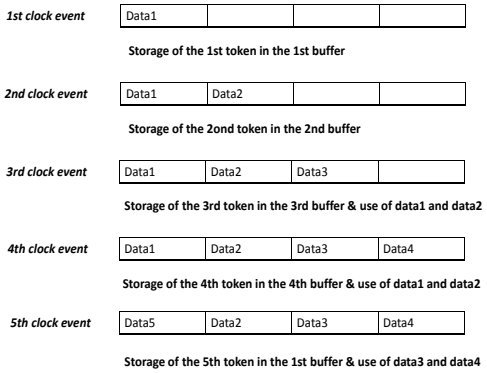

Fig. 11. Ping pong example of a 4-buffer size memory management

Timing performances have considerably increased. Other optimizations can be added by treatment anticipation but they have not been added beacause in that case the design would be a low level one.

A reverse Hadamard block was added for validation. The whole design was compiled with Orcc to obtain the C code of the actors. C codes 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 input as presented in figure 12.

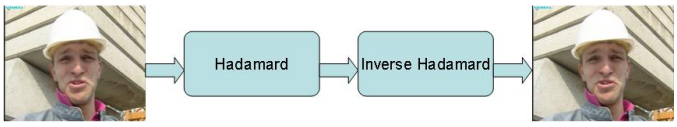

Fig. 12. Software validation

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. Figure 13 shows the example of a “max 2x2” actor in high level description with the “repeat” and the “foreach” loops. This actor is translated to low level one as presented in figure 14.

Thus, we obtained a dataflow implementation of the LAR baseline.

The importance of our approach is to avoid the OpenDF validation of the classic method. In a that method, we used to develop the RVC-CAL codes and add actors for data generation and display. The actor of data generation is composed of a table containing the input image pixel values and some

```
actor max2x2()  uint(size=8) IN ==> uint(size=8) OUT:
  max2x2: action IN:[input] repeat 4 ==> OUT:[out]
  var
    int out:= 0
  do
    foreach int i in Integers(0, 3) do
      out:= if input[i]>out then input[i] else out end;
    end
  end

end
```

Fig. 13. High level RVC-CAL example

```
actor max2x2()  int(size=9) IN ==> int(size=9) OUT:

 int(size=5) cpt:= 0;
 int(size=9) max:= 0;

 init: action IN: [in0] ==>
 do
  max:= in0 ;
 end

 compare : action IN: [in0]  ==>
 do
  if max < in0 then
   max:=in0;
  end
  cpt:= cpt+1;
 end

 send : action ==> OUT: [ max ]
 guard cpt = 3
 do
  cpt := 0 ;
 end

 schedule fsm init :
   init    ( init    ) --> compare;
   compare ( compare ) --> compare;
   compare ( send    ) --> init;
 end

 priority
   send>compare ;
 end

end
```

Fig. 14. High level RVC-CAL example

actions to consecutively put these values in the input port of the design. The design output data can be displayed using the "println()" function. Validation is consequently a tough and relentless value comparison. The use of C compilers allows us to use images and videos for the test and we can have more information about an error when we have both data values and image display.

B. Results

The HDL project manager environment used is Xilinx ISE Foundation 11.1 and the hardware simulation tool is ISE simulator (Full version). We developped 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	1.437/30.720 (4%)
Occupied Slices	2.027/15.360 (13%)
4 input LUTs	3.637/30.720 (11%)
FIFO16/RAMB16s	26/3192 (13%)
Bonded IOBs	99/448 (22%)

TABLE I
CONSUMPTION FOR 352X288 IMAGE SIZE

The time synthesis performances are mentioned in table II Optimization solutions are in development to decrease the

Criterion	352x288
Output frequency(MHz)	22.4
maximum frequency(MHz)	81.4
Latency(μs)	79.4
processing time(ms)	4.5
Minimum input arrival time before clock(ns)	12.134
Maximum output required time after clock(ns)	8.188
Maximum combinational path delay(ns)	5.083

TABLE II
TIMING RESULTS

latency and consequently increase the frequency. In terms of development time, the whole design took about 70 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 where the debug and the validation are easier and faster. The most disturbing part of the flow was the manual transformation of the RVC-CAL from high to low level. This can be explained by the fact the the code is longer and consequently harder to debug because of the inaccurate feedback of Cal2HDL. We are currently looking for solutions to automate this step. This task may be achieved by improving Cal2HDL Java source code or by using the intermediate representation of Orcc. The second case seems to be more feasible. 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.

V. RELATED WORKS

Ihab Amer, in [14], proposed multi-granular RVC tool libraries to synthesize efficient software or hardware implementations from high-level specifications.

In [15], Jani Boutellier shows multiprocessor scheduling of dataflow models within the RVC framework.

By mixing hardware and software generation from RVC-CAL, Richard Thavot presents in [16] a methodology for co-designing complex interfaces systems.

Using the intermediate representation of Orcc, Nicolas Siret is performing an efficient VHDL backend for Orcc.

Johan Eker presents in [17] multicore scheduling issues for Ericsson mobile platforms using RVC-CAL specifications.

VI. 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 LAR coder baseline. This transform implementation is a part of our work to achieve the implementation of the whole LAR image codec. 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 of 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] 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.

[5] 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/

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

[7] 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.

[8] 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.

[9] O. DEFORGES and M. BABEL, "Lar method: from algorithm to synthesis for an embedded low complexity image coder," *IEEE 3rd International Design and Test Workshop*, 2008.

[10] J. PONCIN, "Utilisation de la transformation de hadamard pour le codage et la compression de signaux d'images," in *Springer-Annals of telecommunications*, 1971, pp. 235–252.

[11] S. W. Golomb, "Run length codings," *IEEE Transactions on Information Theory*, pp. 12(7): 399–401, 1966.

[12] R. F. Rice, "Some practical universal noiseless coding techniques," *Technical Report 79-22*, 1979.

[13] 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.

[14] I. Amer, "Towards multi-granular rvc tool libraries: A case study of cal transformations on the iso/iec mpeg fixed point idct," *DASIP 2009*, 2009.

[15] J. Bouteillier, V. M. Gomez, O. Silven, C. Lucarz, and M. Mattavelli, "Multiprocessor scheduling of dataflow models within the reconfigurable video coding framework," *DASIP 2009*, 2009.

[16] R. Thavot, R. Mosqueron, J. Dubois, and M. Mattavelli, "Hardware synthesis of complex standard interfaces using cal dataflow descriptions."

[17] J. Eker, "Multicore scheduling issues in ericsson mobile platforms," *Parallel Processing Workshops, International Conference on*, vol. 0, p. 1, 2009.