Design of an Embedded Low Complexity Image Coder using CAL language

Khaled JERBI [1], Mickaël RAULET [2], Olivier DEFORGES [2], Mohamed ABID [1]

[1] *CES Lab/ENIS Sfax, Route Sokra KM4 BP W 3038 Sfax, Tunisie*
[2] *IETR/INSA Rennes, 20 Av des Buttes de Coësmes 35 043 Rennes Cedex, France*
Khaled_jerbi@yahoo.fr, Mickael.raulet@insa-rennes.Fr, Olivier.deforges@insa-rennes.fr, Mohamed.abid@enis.rnu.tn

Abstract

The increasing complexity of image codecs and the time to market requires a high level design. Caltrop Actor Language (CAL) is a domain-specific language that provides useful abstractions for dataflow programming with actor. It has been chosen by the ISO/IEC standardization organization in the new MPEG standard called Reconfigurable Video Coding. This framework is adopted to design a multitude of codecs by combining actors. We present in this paper the specification and synthesis of the image coder LAR (Locally adaptive resolution) using the CAL framework. An HDL description and generation tools are used. The results show that such a high level design is possible. The quality of the resulting decoder implementation turns out to be better than that of a VHDL reference design.

In the following, the main parts of the LAR coder will be presented; we will introduce the basic notions of the CAL language and its infrastructure (edition, simulation and HDL synthesis tools) and the results will be discussed.

1. Introduction

MPEG has produced several video coding standards such as MPEG-1, MPEG-2, MPEG-4 Video, AVC and SVC. However, the past monolithic specification of such standards (usually in the form of C/C++ programs) lacks flexibility and does not allow to use the combination of coding algorithms from different standards enabling to achieve specific design or performance trade-offs and thus fill, case by case, the requirements of specific applications.
So as to overcome the limitations intrinsic of specifying codecs algorithms by using monolithic imperative code, Caltrop Actor Language (CAL) [1], has been chosen by the ISO/IEC standardization organization in the new MPEG standard called Reconfigurable Video Coding (RVC) (ISO/IEC 23001-4 and 23002-4). RVC is a framework allowing users to define a multitude of different codecs, by combining together actors (called coding tools in RVC) from the MPEG standard library written in CAL [2], that contains video technology from all existing MPEG video past standards (i.e. MPEG- 2, MPEG- 4, etc.). CAL is used to provide the reference software for all coding tools of the entire library. The originality of this paper is the application of the CAL and its associated tools on an image coder. The architectures design was more much faster comparing with an HDL development, while leading to efficient implementation solutions.

In the next part, the state of the art will be presented. The part 3 will briefly introduce the most important original points of the LAR coder. In the part 4 the CAL language and its associated tools will be detailed. Part 5 will show the developed architectures of the Flat LAR and the obtained results. The paper will be closed by presenting the conclusion and the perspectives of further works.

2. State of the art

The use of the CAL language and CAL2HDL tool for HDL generation has increased since the publication of the interesting primary implementation results [3]. Xilinx was the first company adopting the CAL approach to generate IPs for complex applications [7]. In MPEG community, the RVC is currently under development as the part of MPEG-B and MPEG-C standards. 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. RVC provides both a normative standard library of FUs and a set of decoder descriptions expressed as networks of FUs. Such a representation is modular and helps the reconfiguration of a decoder by modifying the topology of the network. RVC mainly focuses on reusability by allowing decoder descriptions to contain common FUs across standards. The MPEG-4 SP decoder has already been developed, tested and compared with the same decoder developed directly in HDL for more details of this comparison see [3].

In Europe, some research laboratories are using the framework to design image processing co-processors. The main application is a smart camera for bar code reading [4].

In the INSA of Rennes (France), the IETR group laboratory is working on an original co-design tool of image processing architectures generation called PREESM [5]. This tool aims at a mixed (HW and SW) architecture generation that is why an equivalent tool to CAL2HDL called CAL2C has been developed for an automatic transformation from CAL to C and the compiler is actually being optimized. Consequently, starting with a CAL description of an application, PREESM would be able to classify the actors into HW and SW ones and automatically generate an HDL code (for HW actors) and a C code (for SW actors).

Based on the interesting results of the CAL and associated tools approach, we proposed to apply it on the LAR coder: an original image coder developed in the IETR laboratory. In addition, some LAR functions' architectures have been implemented and tested using a direct VHDL development. That is why we aimed at comparing the results with those using a higher level conception method (CAL language development and automatic transformations). The principle of the LAR coder is presented in the following.

3. LAR coder

The LAR method [6] relies on the idea that the resolution can be locally adapted to activity: when local luminance remains uniform, the resolution can be low, whereas for high activity areas, good image representation requires a finer resolution. The second principle in the LAR method relies on the fact that an image can be seen as the superposition of two complementary components:

$$I = \bar{I} + \underbrace{(I - \bar{I})}_{E} \qquad (1)$$

$\bar{I}$ represents the image global information (local average value for instance), estimated on a given support, and E is the local variation around it (say local texture). The range of E is dependent on two main factors :

1) the image local activity,

2) the support dimension of $\bar{I}$.

Moreover, if we assume that an image can be roughly seen as the composition of homogeneous areas and contours, then the sole adaptation of the support leads to low dynamic range of E in uniforms areas. On the other hand, E dynamic range will be wide on contours as soon as the support of $\bar{I}$ is superior to one pixel.

In connection with the previous remarks, the main concepts of the LAR method rely on a two layers codec: the first layer called FLAT LAR represents the global information, whereas the second stage, called error spectral coder, provides the local texture. By construction, the LAR method enables two layers of scalability. Figure 1 shows the associated global scheme.

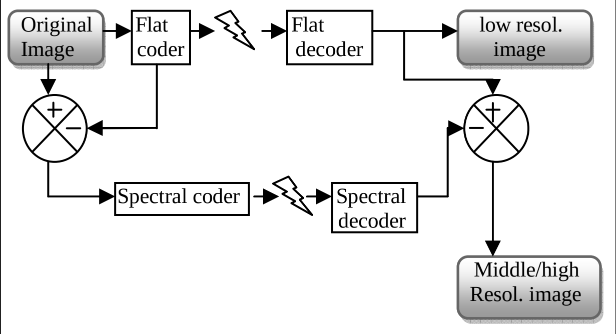

Fig.1. Global LAR scheme: codecs FLAT + spectral layers

This work mainly focuses on the synthesis of the first layer (the Flat LAR). The expression "FLAR" means that the representation of $\bar{I}$ in first layer is performed by local flat approximations. As the objective of this FLAT codec is only to compress the global image information, it clearly addresses high compression rates. The relative image representation aims to distinguish between contours and the remaining of the image, and to adapt the support of $\bar{I}$ such as the rebuilt image remains visually acceptable and presents reduced error E especially in uniforms areas. The support takes here the shape of squared blocks. The global scheme of the FLAT coder is given on figure 2. It relies on a variable block size representation of the image, in which blocks are filled by their mean gray level value. All data produced by the different processing steps (partitioning, prediction errors ...) are further losslessly compressed by a low complexity entropy coder.

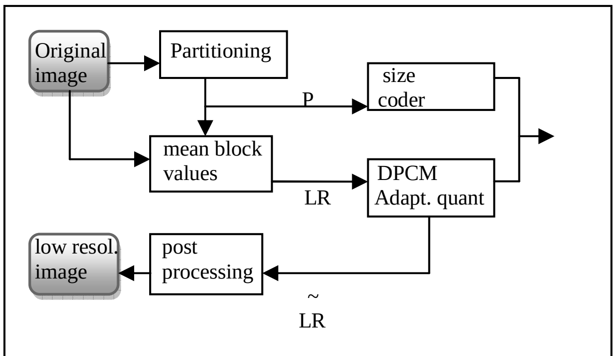

Fig.2. FLAT coder scheme

After LAR presentation, the next part will present the CAL language used for the description of the Flat LAR functions.

4. CAL language and tools

The CAL language (Caltrop Actor Language) has been created in the Ptolemy II project in Berkeley's university. It is based on actors oriented dataflow specification. This specification is not dedicated for a unique language or platform. The main advantage of using this language is that it is placed between the C and the VHDL languages and two open source softwares have been developed to automate the transformation of the CAL code to C (CAL2C) and to VHDL&VERILOG (CAL2HDL). CAL describes algorithms by using a set of encapsulated dataflow components called actors communicating with each others. The topology of the connections between actors input and output ports constitute what is called a network of actors. It is expressed by using an XML dialect known as network language (NL) that also includes the possibility to includes attributes (i.e. parameters) that may be different for the instantiation of the same (parametric) actor in a network of actors. In addition, some researchers from the IETR developed plug-ins in the ECLIPSE tool as Graphiti (http://sourceforge.net/projects/graphiti-editor), which allows the modeling and the connection of the actors with graphic instances to obtain the required networks.

4.1. The language's main notions

In the following, the CAL main notions will be detailed:

4.1.1. Actor: An actor is a parametric entity with inputs, outputs and an internal state. It cannot change the state of another actor in the network but it can only communicate by exchanging tokens throw connected inputs/outputs that is why we consider CAL as a dataflow language. The execution of an actor is based on the execution of elementary functions called actions. The modeling of the actor states can be done using a finite state machine with the appropriate priorities if necessary. The general form of an actor is shown below:

```
actor mem_mgt (generics) int input ==> int output:
action1
action2
…
end
```

4.1.2. Action: While executing an action some tokens are consumed and others are produced independently from the current state of the actor. The execution of an action can be controlled by a finite state machine or by a specified condition using the "guard" syntax or both of them. The "guard" is an expression to test the value of an input token or a local variable.

4.1.3. Priority: If more than one action can be executed at the same time, it is very important to define the priority between them so the notion of priority has been introduced in the language. This notion is very important for the finite state machines in case of concurrent actions.

4.1.4. Finite state machine: The actor functioning can be scheduled using a finite state machine. The required information is: the initial state, the action that changes the current state and the next state. If more than one action can change a state than a priority is advised.

4.2 CAL associated tools

CAL is supported by a portable interpreter infrastructure called Open Data Flow environment (OPENDF) [2].

4.2.1. Simulation: In OPENDF we can simulate a hierarchical network of actors. We can use the Graphiti editor (http://sourceforge.net/projects/graphiti-editor) to create networks of actors in DDL format.

4.2.2. Hardware synthesis with CAL2HDL: The CAL to HDL conversion is done using the CAL2HDL tool which makes the following conversion steps: After parsing, CAL actors are instantiated with the actual values for their formal parameters. The result is an XML representation of the actor which is then precompiled (transformation and analysis steps, including constant propagation, type inference and type checking, analysis of data flow through variables...), represented as a sequential program in static single assignment (SSA) form (making explicit the data dependencies between parts of the program) [7]. Then follows the synthesis stage which turns the SSA threads into a web of circuits built from a set of basic operators (arithmetic, logic, flow control, memory accesses and the like). The synthesis stage can also be given directives driving the unrolling of loops, or the insertion of registers to improve the maximal clock rate of the generated circuit. The final result is a Verilog file containing the circuit implementing the actor, and exposing asynchronous handshake style interfaces for each of its ports. These can be connected either back-to-back or using FIFO buffers into complete systems. The FIFO buffers can be synchronous or asynchronous, making it easy to support multi-clock-domain dataflow designs.

The connection between actors and FIFOs is generated using an Intellectual Property developed with Xilinx's Actor/DataFlow compiler which generates I/O conform to a protocol that is highly compatible with standard FIFO interfaces with First Word Fall Through (FWFT). Each interface is composed of a small number of individual signals providing a simple handshake protocol which ensures reliable, in-order, transfer of data values (tokens) across the interface. These signals and their definition in table 1.

Signal	Description	Direction	Width
DATA	N-bit data signal containing token value	Producer to consumer	N bits
SEND	1-bit value indicating that the value on DATA is valid	Producer to consumer	1 bit
ACK	1-bit acknowledge signal indicating that the token on DATA is consumed	Consumer to producer	1 bit
RDY	1-bit signal if asserted the consumer is indicating that an ACK will be provided immediately	Consumer to producer	1 bit

Table1: Interface signals definitions

The incorporation of Xilinx's Actor/DataFlow compiler generated IP [8] into a system at the electrical level requires very little logic. As shown below, any system which is capable of communicating via traditional FIFO memory protocols can be connected to the generated IP by means of a small number of combinatorial gates. The figure 3 presents the signal level view.

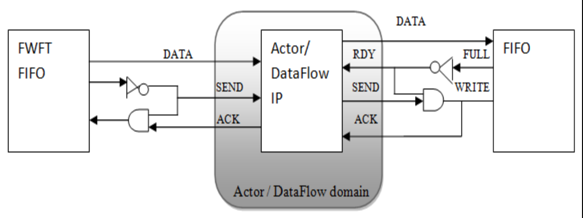

Fig.3. Signal level view

The LAR coder and the CAL framework are now presented. In the following we will detail the functioning of the Flat LAR, present the designed architectures and show the obtained results.

5. The Flat LAR design and test

The objective was to achieve the flat LAR architecture using the CAL language. We supposed that we read the original image pixel per pixel. As the partitioning and the mean block values treatments are independent, we proposed to design them in parallel in a first stage which precedes a selection stage and the DPCM one.

5.1. Flat LAR Principle

As presented previously, the Flat LAR is composed of 3 main parts: the partitioning, the block mean value and the DPCM. These parts are detailed in the following.

5.1.1. Partitioning and block mean value:

-Partitioning: Every system relying on a variable block size representation of images induce an activity criterion (or homogeneousness) and a particular partition topology [9]. In the following, we consider the Quadtree partition $P^{[Nmax \ldots Nmin]}$, in which N_{max} and N_{min} respectively represent the maximal and minimal allowed block sizes, expressed as power of 2. I(x , y) denotes a pixel in the image with coordinates (x , y), and $I(b^N(i, j))$ is the block $b^N(i , j)$ in I such as :

$$b^N(i,j) = \{ (x,y) \in N_x \times N_y \mid N \times i \leq x < N \times (i+1), \quad N \times j \leq x < N \times (j+1) \} \quad (2)$$

Among all existing coding techniques, some of them also consider an image partitioning. For instance, the intra mode of MPEG4-AVC enables two sizes for blocks, i.e. 4 and 16.
It can be also seen as a $P^{[16,4]}$ Quadtree partition. The block size in AVC is selected according to distortion/rate criteria, from a PSNR point of view

[10]. Other methods involve a finer partitioning such as tree decomposition solutions: the decomposition starts at the highest level of the tree (maximal size), and a node is decomposed into four sons since the activity threshold is exceeded. Several homogeneousness criteria have been proposed, [11], [12], but in most of cases they rely on the estimation of L_1 or L_2 norms distance between a given block and its sons. In our approach, we have adopted a different criterion, as the notion of activity is closely here associated to the presence of contours or not. Thus, the activity estimation is performed by a morphological gradient (difference between the maximal and minimal values inside a block). We consider the Quadtree partition $P^{[Nmax...Nmin]}$.
The sizes image directly provides a coarse image segmentation map: blocks of size Nmin are mainly located upon edges and highly textures areas of the image. We will show in the following that this feature enables an adapted quantization in the coding process.
Mean block value: A low resolution image LR is reconstructed by representing each block by its mean value. Consequently, for each pixel p(x,y) we get :

$$LR(x,y) = \frac{1}{N}\sum_{K=0}^{N-1}\sum_{m=0}^{N-1} I(\left[\frac{x}{N}\right] * N + k\left[\frac{y}{N}\right] * N + m)$$

With N=size(x,y) (3)

5.1.2. Block mean value encoding by DPCM approach:

- *Block mean value quantization:* Compression techniques relying on distortion/rate optimization try to find the best compromise between the coding cost and global errors, but only from a PSNR or MSE point of view, without any consideration about the human vision. Nevertheless, experimentations have demonstrated that this human eye is much less sensitive to luminance and chrominance variations in areas such as edges (high visual frequencies) than in uniform areas (low visual frequencies). This principle is simply used in our coding scheme by performing a quantization adapted to the block size. Let qN be the quantization step for blocks of size N. Then using a set of values, such as qN= (qN / 2) / 2, leads to a similar visual quality in the image.

- *Block gray level mean value prediction:* The block mean value encoding is directly realized in the spatial domain, by a DPCM (Differential Pulse Coding Modulation) approach. This choice has been firstly motivated by the simplicity of the coding process, requiring only one regular image scan. Secondly, the block representation provides an interesting a priori about activity areas, which can be useful to adapt prediction. Figure 4 shows the principle of the DPCM.

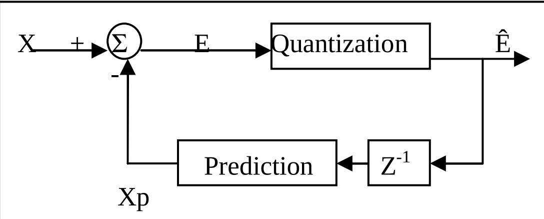

Fig. 4. The DPCM principle

If we consider the 3 neighbors of the X pixel as presented in figure 5 then the predictor is given by (4):

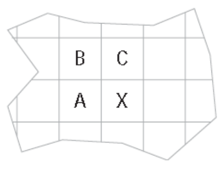

Fig. 5. Neighbor pixels for the DPCM

$$Xp = \begin{cases} A \text{ if } |B-C| < |A-B| \\ C \text{ Else} \end{cases} \qquad (4)$$

5.2 Existing architecture

In the beginning, the LAR coder has been developed in C language and the Flat LAR was implemented on TI TMSC6416 DSP running at 400 MHZ with 1 MO of internal memory [9]. Then, aiming at an FPGA implementation, a dedicated architecture with limited latency and memory requirement was designed in VHDL (figure 6). A synchronous system with a data flow rate of one pixel incoming and out coming per each clock cycle was imposed. The whole architecture was finally improved by pipelining the DPCM functioning. The results are summarized in table 2 below:

	Image size	
	64x64	352x288
Internal memory (octet)	684	3470
4inputs LUTs for logical operations	1166	2463
Frequency (MHZ)	45.8	33
Processing time (ms)	0.09	3.1
Latency time (µs)	18.6	75

Table2: Synthesis results of the existing architecture

5.3 Achieved architecture

The different actors realizing the Flat LAR functions have been developed in CAL language. Every stage of the coder was simulated independently. When all stages were validated, we worked on decreasing the complexity of the computing actors by separating the treatment FUs from the data management FUs through the development of two data management actors that receive the pixels values, store them and send them in 2X2 blocks. We used this separation because the estimation results of the 2X2 blocks are used for the estimation of the 4X4 and 8X8 blocks. Finally we assembled all the actors in a global network using Graphiti as shown in figure 7.

In the first stage, the min, max and mean actors will compute and send the results without saving. The superposition of 3 max, min and sum actors (figure 7) is explained by the fact of imposing an architecture using the partial results of some actors to simplify others. The first max, for example, produces the maxes of the all the 2X2 blocks of the image. So to get the max of a 4X4 block, it is easier to compute the max of its four 2X2 blocks rather than computing the max of all its 16 values (idem for min and sum actors).

In the second stage, an actor receives the means and the size images to generate the mean block image by placing the means in the adequate blocks as shown in the following example where a) is a size image, b) is the RAM containing the means of all the 2X2 and 4X4 blocks and c) is the resulting mean block image:

4	4	4	4	2	2	2	2
4	4	4	4	2	2	2	2
4	4	4	4	2	2	2	2
4	4	4	4	2	2	2	2

a) size image

Means 2X2	14	22	50	9	12	27	43	23
Means 4X4	65	34	-	-	-	-	-	-

b) Mean values table

65	65	65	65	12	12	27	27
65	65	65	65	12	12	27	27
65	65	65	65	43	43	23	23
65	65	65	65	43	43	23	23

C) Mean block image

In the 3rd stage, the DPCM actor gets the size and the mean block images as inputs and produces the quantified errors and the quantified image using the algorithm presented in 5.1.2.

5.4. Experiments and Results

The achieved architecture has been simulated with the OPENDF simulator for image sizes of 64x64 and 352x288. This operation took only 3 minutes.

After the simulation, some lines of the CAL codes have been modified to get synthesizable ones. These modifications concerned especially the multiplication and the division operations transformed into left and right shifts. We also added generic parameters for the input and output sizes for further dynamic configurations. The resulting instances were successfully synthesized into HDL codes using CAL2HDL. The HDL project manager environment used was Xilinx ISE Foundation10.1 and the hardware simulation tools were ModelSim Xilinx Edition 6.3 and ISE simulator (Full version). We developed the test benches manually by initializing the different signals shown in table 1 and figure 3 and by generating the stimulus values corresponding to the sizes.

After compilation, simulation, RTL synthesis and place & route on the virtex4: xc4vlx15, package sf363, speed -12, we obtained the results presented in table 3:

Criteria	**consumption**		**percentage**	
	64x64	**352x288**	**64x64**	**352x288**
Number of BUFGs	1/32	1/32	3%	3%
Number of External IOBs	55/240	55/240	22%	22%
Number of LOCed IOBs	0/55	0/55	0%	0%
Number of Slice Registers	5812/6144	594/6144	94%	96%
Number of SLICEMs	428/3072	668/3072	13%	21%
Number of 4 input LUTs	7423/12288	7946/12288	60%	64%

Table3: Device Utilization Summary

Concerning the time synthesis report we mention that the maximum frequency obtained is 136.983MHZ for a 64x64 image corresponding to a minimum period of 7.30 ns and 130.1 for a 253x288 image corresponding to a minimum period of 7.6 ns.

A comparison between the CAL method and the VHDL one will be detailed in table 4. In this table we will put the output frequency which is a more significant criterion than the circuit functioning frequency.

	VHDL		CAL	
	64x64	352x288	64x64	352x288
Internal memory (octet)	684	3470	3168	13088
4inputs LUTs for logical operations	1166	2463	7423	7423
Frequency (MHZ)	45.8	33	8.5	8
Processing time (ms)	0.09	3.1	0.33	11.7
Latency time (µs)	18.6	75	26.6	27.8

Table4: CAL & VHDL performances comparison

We expected such difference between the results but our obtained performances remain acceptable if we consider the area consumption and the advantages of a high level design in terms of conception time and functions management.

6. Conclusion and perspectives

The central points of this paper can be summarized as follows:

Considering the new MPEG RVC coding method, the CAL language was the most adequate candidate. The realized works were very successful in terms of conception time and implementation results. Our task was to apply the approach on the LAR coder, previously developed in VHDL, to verify if the results of a higher level conception method are acceptable.

The paper presented: The state of the art, the LAR codec and its functioning principle, the CAL language and its associated tools, the realized architectures and the implementation time and area results.

These results were very encouraging since they are near those obtained with an optimized code developed directly in VHDL. There are many optimization solutions in progress to improve the results especially in terms of area consumption. Architecture in progress will pipeline the different steps of the DPCM (estimation, error quantification and quantified image filling) for a better functioning. This architecture will be the same used to develop the Flat LAR directly in VHDL.

References

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

[2] Shuvra S. Bhattacharyya, Gordon Brebner, Jörn W. Janneck, Johan Eker, Carl von Platen, Marco Mattavelli and Mickaël Raulet, "OpenDF – A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems" in Proceedings of *the Swedish Workshop on Multi-Core Computing*, November, Ronneby, Sweden, 2008.

[3] Jörn W. Janneck, "Notes on an actor language" *in 7th Ptolemy Miniconference*, 13 February 2007.

[4] Richard Thavot, Romuald Mosqueron, Mohammad Alisafaee, Christophe Lucarz, Christophe Lucarz and Julien Dubois, "Dataflow design of a co-processor architecture for image processing" in Proceedings of *the Workshop on Design and Architectures for Signal and Image Processing (DASIP 2008)*, Bruxelles , Belgium, November 2008.

[5] Christophe Lucarz,Marco Mattavelli, Matthieu Wipliez, Ghislain Roquier, Matthieu Wipliez, Mickaël Raulet, Jörn W. Janneck , Ian D. Miller and Dave B. Parlour "Dataflow/Actor-Oriented language for the design of complex signal processing systems" in Proceedings of the *Workshop on Design and Architectures for Signal and Image Processing (DASIP 2008)*, Bruxelles , Belgium, November 2008.

[6] Déforges O., Babel M., Bédat L., Ronsin J., "[09/07/2009 11:06:08] Mickael Raulet: fulltext access Color LAR codec: a color image representation and compression scheme based on local resolution adjustment and self-extracting region representation", *IEEE Transactions on Circuits and Systems for Video Technology 17, 8 (2007) 974-987.*

[7] 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*, 2008.

[8] Xilinx, "Xilinx Actor/DataFlow interfacing and protocol", *ASTG technical Memo Programming Solutions Group*, Xilinx, May 27th 2008.

[9] Olivier Déforges and Marie Babel "LAR method: from algorithm to synthesis for an embedded low complexity image coder" in Proceedings of *the 2008 IEEE 3rd International Design and Test Workshop, IDT'08*, Monastir: Tunisia (2008).

[10] H264 MPEG-4 10 AVC, "Joint committee draft (cd)," *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEGn 3rd Meeting: Fairfax*, Virginia, USA, May 2002.

[11] C.A. Shaffer and H. Samet, "Optimal quadtree construction algorithms," *Computer Vision, Graphics, Image processing*, vol. 37, October 1987.

[12] P. Strobac, "Tree-structured scene adaptive coder," *IEEE Trans. On Communication*, vol. 38, no. 4, April 1990.

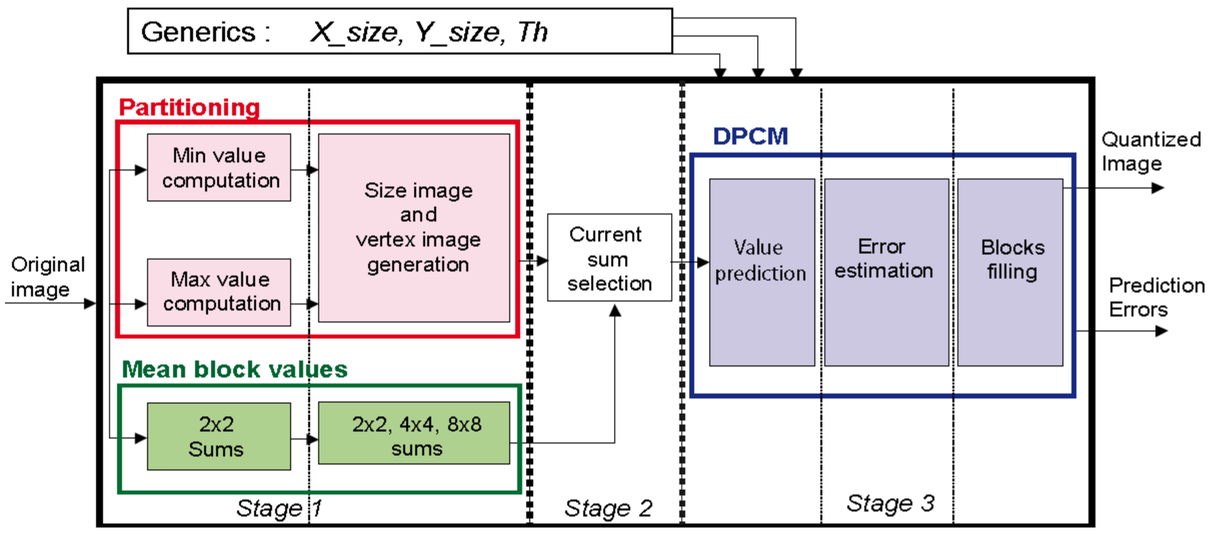

Fig. 6. Flat LAR: proposed architecture

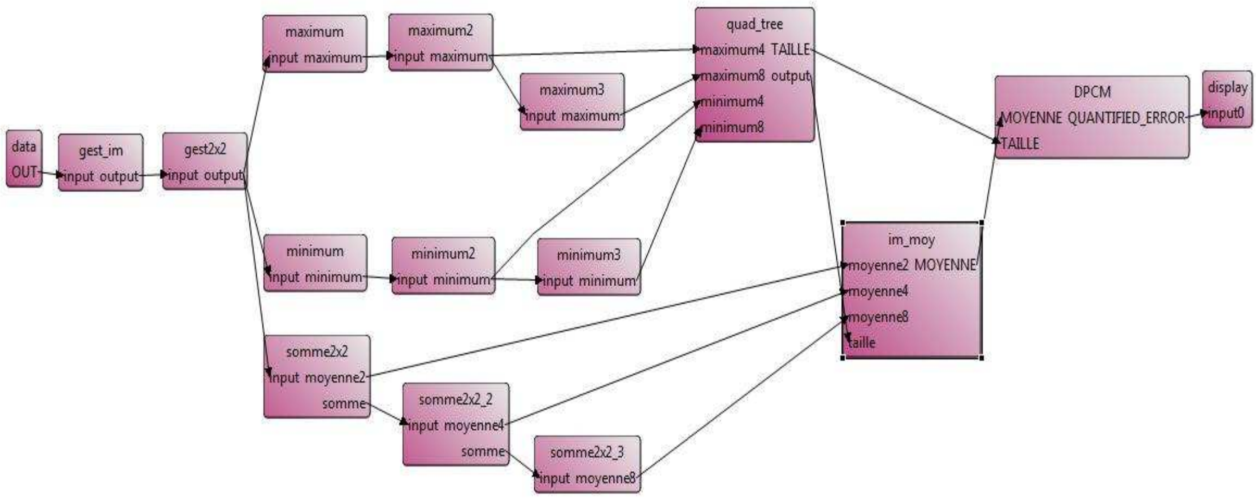

Fig. 7. Flat LAR: Realized architecture