International Conference on Control, Engineering & Information Technology (CEIT'13)
Proceedings Engineering & Technology - Vol.2, pp. 211-216, 2013

Performance mesurement of multiprocessor architectures on FPGA(case study: 3D, MPEG-2)

Kais LOUKIL[#1], Faten BELLAKHDHAR[#2], Niez BRADAI[*3], Mohamed ABID[#4]

[#]*Computer Embedded System,* National Engineering School of SFAX, University of SFAX
Soukra city, Sfax 3038, Tunisia

[1]kais_loukil@yahoo.fr
[2]belfaten@yahoo.fr
[3]bradai_niez@yahoo.fr
[4]mohamed.abid@enis.rnu.tn

Abstract— **Nowadays, developers are more and more leaning towards multiprocessor embedded processors in their systems designs as they need further performance. In this context, our work aimed at prototyping several multiprocessor architectural solutions on FPGA using the Altera development environment and implementing two multimedia applications: the MPEG-2 decoder and the 3D synthesis.**
The MPEG-2 decoder is successfully implemented on a dual-core architecture allowing the decrease of the execution time from 1.45 sec to 0. 905 sec. Besides, the 3D synthesis implementation on an architecture consisting of four core processors adhered to the real time constraints by providing a rate of 27 frames per second.

Keywords-Multiprocessor;Performance; reconfigurable; SoC

I. INTRODUCTION

Multiprocessor devices are driving progressively into embedded applications. Single-core processors and the performance imperative of Moore's Law may be approaching an upper limit in terms of adding increasing processing power simply by increasing clock speeds. Consequently, embedded designers have turned, instead, to multiprocessors in order to achieve performance gains. Multiprocessor technology offers opportunities to improve the processing performance and power efficiency. But in the other hand, it also requires different programming models from those used for uniprocessors. The real challenge currently is the ability to develop the software within a reasonable time scale; the lack of standards and integrated tools makes the software tasks much more difficult [3].

There is a great deal of opportunities in the embedded multi-core market, however, it is evident for most observers that a major gap currently exists between multi-core silicon and software enabled to take advantage of the available performance. In this context, our work consists in prototyping several multiprocessor architectural solutions by the migration of single core designs to multiprocessor architectures. This work will be validated by implementing two multimedia applications: the MPEG-2 decoder and the 3-D synthesis, on FPGA using different implementations of the source code.

For each application, we started by implementing the code on a single standard hardware architecture, then we tried to transform and rewrite certain functions of the source code in order to adapt the software to fit these multiprocessor architectures. This work will be followed by a performance evaluation of these prototypes including the total execution time, the surface and power consumption.

As prototyping platform we have used the technology and the development environment ALTERA, something that has allowed us to identify, and by the way to overcome and resolve several limitations of this environment.

This paper is organized into three sections structured as follows: The first section is dedicated to introduce the state of the art of multiprocessor processors, evoking the main reasons of this tendency and presenting some examples of multiprocessor processors in the embedded market. The second section provides an overview of the MPEG-2 standard and the 3-D Synthesis; the multimedia applications that served for prototyping. The third section focuses on the prototypes validation; it details the different approaches followed throughout the implementation phase and presents the results and the performance measurement of our multiprocessor architectures.

II. STATE OF THE ART

Actually, developers are more and more leaning towards multi-core embedded processors in their systems design as they need further performance. In the last 10 years, to meet performance requirements, processors are faster mainly due to increasing clock frequencies or more complex architectures. Running smaller transistors at faster speeds has driven exponential increases in performance but the challenge is that each transistor on a chip consumes power and produces heat and the faster the transistors are clocked, the more heat they generate [4].

Decreasing a processor's frequency and voltage leads to an important reduction of its total power requirements, even small speed reductions can make a big difference. Semiconductor manufacturers have figured out that the way forward is to build processors running at both lower frequencies and voltages, and additionally to integrate two or more of these processing cores on a single die [5, 8]. Thus, industry is currently turning from increased frequency to parallelism. The power efficiency inherent in dividing work among multiple processor cores on

one die allows continuing dramatic increases in performance while reducing power consumption and heat dissipation. In fact, multiprocessor processors can execute instructions in parallel, which means multiple separate instruction threads can be processed at the same time. Hence, chip companies have turned to multiprocessor designs in recent years to bring the power of parallel processing to embedded systems. Currently, all processor vendors have multiprocessor processors on their product road maps, and many have already released products. There are two distinct segments with distinctly different approaches that have emerged: general-purpose multi-core processors and application-focused multi-core processors [10].

General-purpose multi-core processors represents processors with multiple, usually homogeneous, cores, in which any (or all) of the cores may be called upon and used to provide the processing needs within an application. In contrast, application-focused multi-core processors provide different cores for different pieces of an application. For example, one core may process audio and while another processes video. Cores may be homogeneous or heterogeneous, depending on the methodology used in the processor's design. Note that these different segments of the embedded multi-core market utilize very different approaches, and target different kinds of applications. It is very important for users to understand each approach, and which one is best suited for their particular application [6]. The first multi-core CPUs offered to the embedded market were released in late 2006, in the form of dual core processors [1]. In 2007, multi-core product portfolios have been expanded and new suppliers have entered the market, which is projected to grow significantly.

III. Parallel computer Classical Taxonomy

Currently, the most popular nomenclature for the classification of computer architectures is that proposed by Flynn that chose not to examine the explicit structure of the machine, but rather how instructions and data flow run through it. Specifically, the taxonomy identifies whether there are single or multiple 'streams' for data and for instructions [7, 9]. The term 'stream' refers to a sequence of either instructions or data operated on by the computer. Depending on whether there is one or several of these streams, we have four classes of computers:

- Single Instruction Stream, Single Data Stream: SISD
- Multiple Instruction Stream, Single Data Stream: MISD
- Single Instruction Stream, Multiple Data Stream: SIMD
- Multiple Instruction Stream, Multiple Data Stream: MIMD

The Fig. 1 illustrates the differences between the four classes.

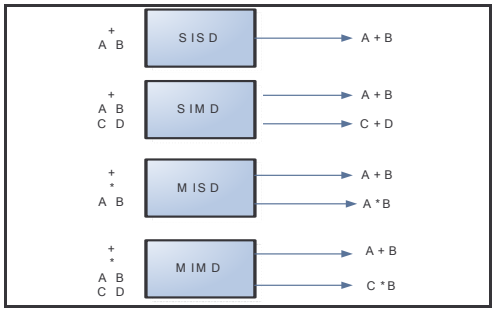

Fig. 1 Potential of the 4 classes

IV. Application examples: MPEG-2 standard & the 3-D Synthesis

After presenting generalities about the multiprocessor architectures and enumerating some of their applications in the embedded domain, we move to detail the theory of MPEG-2 standard and the 3-D synthesis application.

A. *MPEG-2 Overview*

MPEG-2 is a standard for motion video compression and decompression defined by the Motion Pictures Expert Group (MPEG). MPEG-2 extends the basic MPEG-1 to provide compression support for TV quality transmission of digital video. The MPEG-1 and MPEG-2 are already being used in many video applications and their adoption continues to grow rapidly.

B. *3D-Synthesis overview*

A basic 3D_synthesis algorithm takes a 3D object described as a set of triangles and transforms it into 2-dimensional pixel representation. All the necessary operations to display a 3D object reconstitute the graphic pipeline described in Fig. 2.

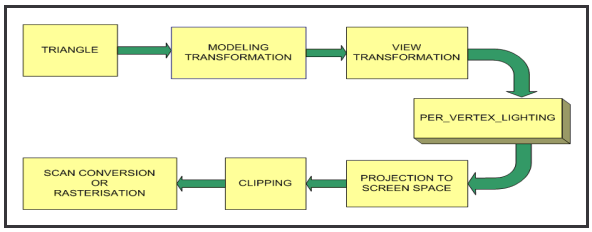

Fig. 2 The 3D-Synthesis graphic pipeline

The choice of the MPEG-2 decoder and the 3D synthesis for our study is based on their continuous adoption for many applications and their real time performance demanding. These applications are characterized by a computational complexity which is much costly for a single processor to achieve real-time performance in software. In this context, our task consists of designing multiprocessor architectures for both applications to enhance their performances compared to their single core implementation and respond to the real-time constraint. In the next section, we will present the results and the performance measurement of the implemented multimedia applications, MPEG2 decoder and 3D synthesis, on multiprocessor architectures.

V. THE MPEG2 DECODER IMPLEMENTATION

A. *The MPEG-2 decoder software*

The basis of our study is an MPEG-2 decoder purely software. This decoder, written in C, is available for free download from the MPEG server.

1) Single-core implementation: The first step is to choose hardware architecture for the decoder implementation. We used the standard hardware example design for the NiosII cycloneII 2c35 development board. In the Nios II environment, we created a software project for the MPEG2 decoder. For all the prototypes, we used a test bit-stream with 3 pictures and resolution of 128x128 pixels.

2) Time execution measurement: The major advantage of measuring with the profiler is that it provides an overview of the entire application. But in the other hand, it is estimation, not an exact representation; of where the CPU time is spent. The most interesting feature of the GNU Profiler is the Call Hierarchy view Fig. 3. It displays the gmon.out call graph data in an easy-to-read tree format. In this view, we can follow easily the function call sequences, which provide greater insight into the timing and the program behavior.

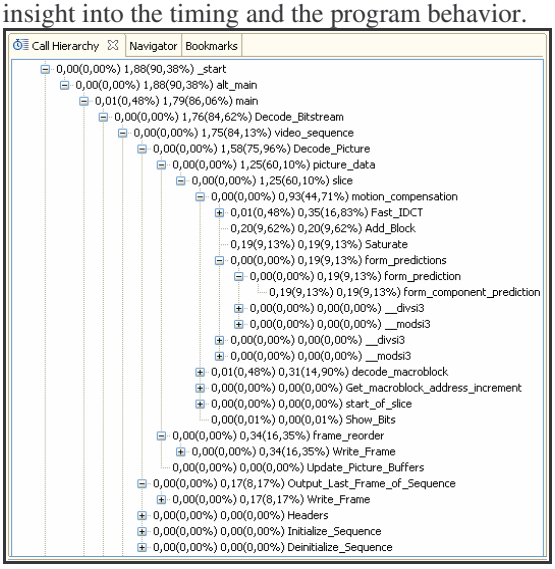

Fig. 3 The call Hierarchy view

After the profiler identifies areas of code that consume lots of CPU cycles, a performance counter can further analyze these functions. With the performance counter, we can accurately measure execution time taken by multiple sections of the code Fig. 4.

Section	%	Time (sec)	Time (clocks)	Occurrences
deco_pic	84.6	0.57462	48843048	3
fram_reor	9.8	0.06660	5661100	3
deco_macro	20.8	0.14111	11994669	192
motion_comp	63.4	0.43106	36640033	192
IDCT	26.5	0.18026	15321924	1152
add_blok	11.8	0.07991	6792718	1152
saturate	15	0.10164	8639597	1152

Fig. 4 Performance report for the primary decoder functions

Enabling the host-based file system, the data traveling between host and target serially through the Altera download cable takes a lot of time nearby 7.231 sec while the total decoding time is 0.679 sec. The host-based file system solution is very expensive in term of time consumption. For the coming implementations, we just consider the decoding time as the resulting execution time.

3) Multiprocessor implementations:

a) First approach: Block level parallelism

- Parallelism sources

Given an MPEG stream, the decoding process performs the five main stages in a sequential order. The only source of parallelism resides on the layered structure of the MPEG-2 bit-stream.

It is a parallelism that exists in the GOP layer, the frame layer and the different levels within a picture: the slice level, the macro-block level and the block level. A previous work [2] presented two parallel implementations of an MPEG-2 decoder; one exploiting parallelism across the GOP (group of picture) in video sequence and the other exploiting slice parallelism within a picture. As there is no way to parallelize at the macroblock layer because macro-block decoding depends on previous macro-blocks for motion compensation, we choose to work at the block level which represents the lowest unit of data at which decoder processes the video stream independently.

- Scenario

To exploit the independency between blocks calculations, the idea was to divide the computation within a macro block on two processor cores working each on the half block number within a single macro block. The motion_compensation function is appropriate to apply this idea as it calls the saturate and fast_idct functions which are time demanding functions and also process at the block level.

- Results

This dual-core architecture didn't enhance performance too much due to the overhead of the communications and data transfer between the two processors. The Fig. 5 shows the performance counter reports for single core implementation and this dual-core implementation. We notice that the time execution of the motion compensation function was reduced by nearby 18% and the total time execution (without using the host) has decreased from 0.679 sec to 0.527 sec. when storing the output files on the host PC, the global time still almost the same Fig. 5 because the host file data traveling between host and target serially through the Altera download cable takes a lot of time (nearby 7.231 sec).

```
--Performance Counter Report--
Total Time: 7.9108 seconds  (672417626 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|montion_comp   | 5.47|    0.43267|       36777364|        192|
+---------------+-----+-----------+---------------+-----------+

Total Time: 7.75949 seconds  (659556847 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|motion_comp    | 4.58|    0.33655|       30205208|        192|
+---------------+-----+-----------+---------------+-----------+
```

Fig. 5Performance counter reports

b) Second approach: Luminance and chrominance:

- Chrominance and luminance independency

In MPEG-2, RGB pixel information is represented as luminance and chrominance components where brightness levels and color information are stored separately. In the 4:2:0 chroma format, a pixel information is represented by a macro-block formed by six 8x8 blocks; four blocks for the luminance and two reserved for the chrominance. Our code works on these blocks independently; indeed, it separates completely between the luminance calculation and chrominance calculation throughout the decoding process. Even at the end of each frame decoding, the resulting luminance and chrominance data are written in different memory areas.

We can assume that the decoder code can be split into two codes; one to handle the luminance calculation and the other to proceed on the chrominance. To assert this assumption, we removed all the code routines related to the chrominance calculations, we compared then the luminance output file (.Y) for each frame with the output files of the original code, we found out that theses files are identical. The same work was done to verify the validity of the chrominance files. The chrominance and luminance independency represents thus a source of parallelism that we can exploit to decrease the overall execution time.

- Time measurement

We used the performance counter to measure the decoding time (without writing the output files on the pc host) which has decreased nearby 40% of the initial measured time. In Fig. 6, the first table represents the performance counter report of a single core decoder and the second table shows the report for the dual-core decoder.

```
Total Time: 1.45862 seconds  (123983114 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|motion_comp    | 64.6|    0.94258|       80119587|        192|
+---------------+-----+-----------+---------------+-----------+

Total Time: 0.905389 seconds  (68458238 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|motion_comp    | 58.7|    0.53267|       456277459|        192|
+---------------+-----+-----------+---------------+-----------+
```

Fig. 6 Performance counter reports

Even after this decoding time reduction, the global execution time still too large because of the host file data option; writing the resulting files on the PC host causes a huge loss of time.

B. The 3D synthesis implementation:

The basis of our study is a 3D synthesis algorithm written in C++. Its input is an ASC file that contains the object name, its vertex coordinates and faces list. This file can be generated by the 3D Studio Max editor. During the rasterization process, the algorithm draws the object first on a virtual screen (a memory zone where the color value for each pixel is stored) then displays the result to the physical screen.

1) . The 3D synthesis' call graph analysis:

Our project aims to transform this multimedia application from a single core design to a multiprocessor architecture. A good understanding of the software code is necessary to achieve this purpose. From the function' call graph Fig. 7, we can follow the code approach

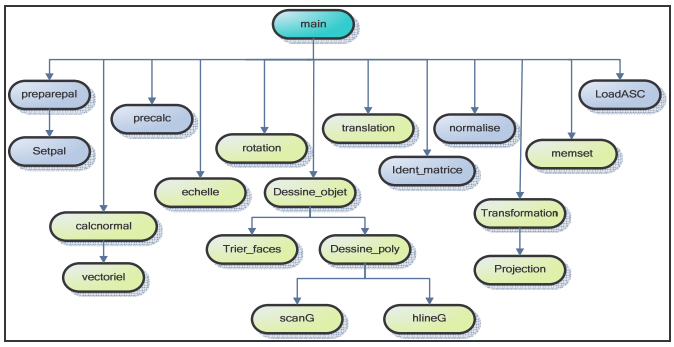

Fig. 7 The 3-D synthesis functions' call graph

2) The 3D algorithm profiling:

The profiling of this application is done by the performance counter. Time consumed by the code principle functions is shown in the Fig. 8. From the timing result report, we notice that the functions ensuring the geometric calculations (echelle, translation, rotation, transformation and calcnormal) are not time demanding; they consume just 12.5% of the global execution time while the dessine_poly function consumes an average of 65% of the global time execution. This time is spent to achieve the rasterization process that requires heavy calculations.

```
finish 3D synthesis code--Performance Counter Report--
Total Time: 43.3803 seconds  (9543669362 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|loadasc        |0.0207|    0.00896|        1970887|          1|
+---------------+-----+-----------+---------------+-----------+
|geo_cal        | 12.5|    5.42374|     1193222758|        360|
+---------------+-----+-----------+---------------+-----------+
|dessine_objet  | 81.5|   36.64073|     8060960045|        360|
+---------------+-----+-----------+---------------+-----------+
|tri            | 6.62|    2.87258|      631967577|        360|
+---------------+-----+-----------+---------------+-----------+
|dess_poly      | 65.1|   28.24638|     6214202617|        360|
+---------------+-----+-----------+---------------+-----------+
|scanline.loop  | 1.81|    0.80984|      178165555|      11489|
+---------------+-----+-----------+---------------+-----------+
|scanG          | 23.9|   10.68443|     2350574784|      11489|
+---------------+-----+-----------+---------------+-----------+
|hlinG          | 40.4|   18.01281|     3962818077|      11489|
+---------------+-----+-----------+---------------+-----------+
|affiche_image  |0.0947|    0.04256|        9364177|        360|
+---------------+-----+-----------+---------------+-----------+
```

Fig. 8 Performance counter report for the 3-D synthesis algorithm

The global execution time for 360 pictures (the rotation angle varies from 0 to 359 degrees) is 43.38 seconds which is

nearby 8 frames per second. Real-time applications of the 3-D synthesis need to respond immediately to user input, and generally need to produce frame rates of at least 20 frames per second (and preferably 60 fps or more). The resulting rate is lower than the average (20frame/sec); the 3D synthesis algorithm performance must be enhanced.

As dessine_poly function is the most time consuming, we should focus on it to figure out if there is any parallelism that may be exploited to decrease its execution time.

3) First multiprocessor architecture approach:

a) Dessine_poly function analysis:

The dessine_poly function works on the shading process; for each visible polygon, it calls first the scanG function to interpolate the color intensity between polygon summits then it calls the hilinG function to accomplish the horizontal interpolation of the color intensity. Finally each calculated color value is stored in the appropriate offset within the virtual screen. This function is called as many times as the number of visible polygon faces within the object. The treatment of this function could be done by two processor cores or more; each one will handle a part of the object polygons.

b) Scenario:

The idea consists in using dual core architecture to implement the 3D synthesis application. The code for each processor is basically the same as the single core approach, just when it comes to the dessine_poly function, the first processor will operate on the half object polygons and the second will achieve the rest of polygons' treatment. The display process is dedicated to the first processor. This processor is responsible for displaying the 3-dimensional object on the VGA monitor each time the virtual screen is completely filled up. Because of the need for mutual communication between processors, a shared memory is used to play the role of a message buffer. At the beginning, the first processor sends the virtual screen address to the second processor; consequently, both processors are able to access concurrently that memory zone in order to fill it up with appropriate data.

c) Time measurement:

From the performance counter report Fig. 9, we notice that the global execution time has decreased to 31% (from 43.38 sec to 30.036 sec); time consumed by the dessine_poly function was reduced by 15.221 sec. This dual core approach has carried out a rate of 12 frames per second, but this rate still lower then what is needed.

```
 finish 3D synthesis code--Performance Counter Report--
Total Time: 30.0362 seconds  (6607961913 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|dessine_poly   | 43.4|   13.02540|     2865588148|        360|
+---------------+-----+-----------+---------------+-----------+
```

Fig. 9 Performance counter report for the first dual core approach

4) Second multiprocessor approach:

This approach takes advantage of the rotation animation while the object display. Like it is previously explained, within the while the same calculations are repeated for each angle incrementation to give the animation effects to the displayed object. This approach was implemented in a first attempt using the dual core architecture. This time, the code won't be split; each processor will execute the entire algorithm independently then the display process will be carried out in an alternative way. The first processor executes the algorithm and displays the object just for the even angles while the second proceeds similarly on the odd angles. The display process is organized by exchanging messages between the processors. This mutual communication is established through a message buffer whose access is protected by a Mutex core. By passing messages, the processors display the object successively in the right order, consequently the object rotation will speed up and global execution time will decrease.

This approach was also applied using three and four core processors. The Table I summarizes the results of the different implementations for the display of 360 frames. With the rate of 27 frames per second, the architecture including four core processors adheres to the real-time constraint that was estimated to 25 frames per second.

TABLE I. THE IMPLEMENTATION RESULTS USING DIFFERENT MULTIPROCESSOR ARCHITECTURES

Core processor number	**The global execution time**		**The frame rate per second**
	sec	***Clock- cycle***	
1	43.3803	9543669362	*8.29*
2	*24.1558*	*5314268946*	*14.90*
3	*17.6867*	*3891072802*	*20.36*
4	*12.9599*	*2851179707*	*27.79*

From the previous results, we can notice that the rate of the total execution time reduction is not the same for the different hardware architectures. Rising the number of processor cores, the total time reduction decreases gradually. In fact, adding more CPUs can geometrically increase the traffic on the shared memory-CPU path and thus decrease the availability of the shared memory to the processors.

5) Performance evaluation:

Throughout this section, we proposed different parallelizing approaches that ground essentially on the code profiling information and parallelism sources.

The multiprocessor execution mode applied for most multiprocessor architectures is the SIMD machine as all the processors execute the same instruction (code) but with different data. Table II summarizes the results of the different implementations achieved during our work. This table shows the total execution time, the surface and the power consumption of each prototype.

TABLE II. COMPARATIVE TABLE OF THE DIFFERENT PROTOTYPES

MPEG-2 Decoder results			
The hardware architecture	*Total logic elements (LE)*	*Power consumption (mW)*	*Execution time (sec)*
a single core (w/ SSRAM)	2890/33216 (9%)	279.14	0.679
a single core (w/ SDRAM)	5020/33216 (15%)	501.43	1.458
a single core + 4 accelerators	14955 / 33216 (45 %)	642.17	0.687
Dual core using SSRAM	6369/33216 (19%)	355.95	0.527
Dual core using SDRAM	7942/33216 (24%)	596.	0.905
3-D synthesis results			
1 core	4192/33216 (13%)	293.09	43.3803
2 cores	8790/33216 (26%)	427.81	*24.1558*
3 cores	12866/33216 (39%)	539.08	*17.6867*
4 cores	16398/33216 (49%)	628.89	*12.9599*

Several factories contribute in defining the performance of a given hardware architecture. In fact, the memory location (on-chip or off-chip memory) and its type (SSRAM, SDRAM or flash memory) influence greatly the architecture performance as they represent the main factors that determine the memory's access latency. As it is shown in table II, there is a huge difference between the performances of the architecture using the SSRAM memory and the one using the SDRAM memory.

Besides, increasing the number of core processors within a design rise in return the total logic elements and the power consumption. In our case, we kept the same frequency for all the prototypes; thus additional processors will increase consequently the power consumption.

The table shows also the performance enhancement (execution time) brought by the multiprocessor architectures for both applications. These improvements are relative to the parallelism sources (partial or total, fine- or coarse - grained parallelism) exploited for each approach.

As a conclusion, we can confirm that the hardware architecture choice depends tightly on the application constraints such as the rapidity, the die surface, the power consumption, the frequency, etc. The combination of optimized hardware architecture with well developed software fitting the design is primordial to achieve better performances.

VI. CONCLUSION

Faced with the race for high frequency processors, disadvantages inherent to the high consumption, heat release and technological limitations led as consequence to the adoption of the multiprocessor architecture solutions. Such trend requires a design methodology on one hand and a development environment, on the other. In this context our work has dealt with a topical subject consisting in prototyping multiprocessor architectures on reconfigurable technology FPGA.

Given the current state, there is a little work in this direction. Consequently, we were brought to seek practical hardware and software solutions, according to the possibilities offered by the Altera platform, to succeed these implementations.

Throughout prototyping, we focused both on hardware and software aspects. First we define the hardware architecture that matches the parallelism approaches. Then, thanks to the Nios II IDE, we went by all the necessary software development tasks for these designs and showed the key issues to establish the communications between processors in such architectures. We can resume that the entire work stages allowed us to control almost all hardware and software steps to design, and test the implementation of single and multiprocessor systems on a reconfigurable target device using the ALTERA development environment.

The MPEG-2 decoder was successfully implemented on a dual-core architecture allowing the decrease of the execution time from 1.45 sec to 0. 905 sec. Besides, the 3-D synthesis implementation on an architecture consisting of four core processors adhered to the real time constraints by providing a rate of 27 frames per second.

References

[1] Eric Heikkila , J. Eric Gulliksen. White paper on: "Multi-core computing in embedded applications: Global Market Opportunity and Requirements Analysis". Venture Development Corporation. August 2007.

[2] Angelos Bilas, Jason Fritts, Jaswinder Pal Singh "Real-Time Parallel MPEG-2 Decoding in Software" 11th International Parallel Processing Symposium 1997

[3] E. O. Kosorukov and M. G. Furugyan "Some algorithms for resource allocation in multiprocessor systems" Moscow University Computational Mathematics and Cybernetics Volume 33, Number 4, 2009

[4] Vahid Kazempour, Alexandra Fedorova and Pouya Alagheband " Performance Implications of Cache Affinity on Multicore Processors" Euro-Par 2008 – Parallel Processing Lecture Notes in Computer Science, 2008, Volume 5168/2008,

[5] Göhringer, D., Becker, J. "High performance reconfigurable multi-processor-based computing on FPGAs" international Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, IPDPSW 2010 ATLANTA (Georgia) USA.

[6] Go□hringer, D.; Hu□bner, M.; Benz, M.; Becker, J "A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip"

[7] Yuan Xie; "Processor Architecture Design Using 3D Integration Technology" VLSI Design, 2010. VLSID '10. 23rd International Conference

[8] Wolf, W. Jerraya, A.A. Martin, G. "Multiprocessor System-on-Chip (MPSoC) Technology" Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions 2008 Volume: 27 Issue: 10

[9] Salih, M.H.; Arshad, M.R.; "Design and implementation of embedded multiprocessor architecture using FPGA " Industrial Electronics & Applications (ISIEA), 2010 IEEE Symposium on

[10] Brandenburg, B.B.; Calandrino, J.M.; Block, A.; Leontyev, H.; Anderson, J.H. "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?" Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE