Self adaptive Augmented Reality systems on FPGA

Tarek FRIKHA, Nader BEN AMOR, Ines BENHLIMA, Kais LOUKIL, Mohamed ABID
CES-Laboratory
Sfax University, National Engineering School of Sfax
Sfax TUNISIA
tarek.1982@gmail.com

Jean-Philippe DIGUET
Lab-STICC
University Bretagne Sud,
Lorient, FRANCE
jean-philippe.diguet@univ-ubs.fr

***Abstract*: Augmented reality (AR) systems emerged and become a very gifted 3D embedded multimedia application. AR consists on adding specific 3D's animations on a video flow. The design and the implementation of such systems on FPGA are complex and difficult. To reduce computational resources that must be carefully used to execute complex application. This execution can be often done in unpredictable environments: it is the major problem to solve. The system architecture must be efficient and flexible enough to adapt system resources to the application requirements and the environment architectures and mobile's constraints. In this paper, we describe our concept of flexible architecture: we have developed IPs to obtain self-adaptive augmented reality systems to implement on FPGA.**

I. INTRODUCTION

THE multimedia embedded applications inflate the computer sciences domain. Watching a HD video or a 3D movie is now possible not only with a 3D TV but also possible with new Smartphone, video games and others.

Thus, the design of multimedia systems applications becomes a major research axis. These applications must have an acceptable quality of service (QoS) despite the limitation of used resources (computing, energy, data transfer…) and the environmental external fluctuations (noise, dust, vibrations…).

To design an efficient embedded product it's important to maximize the product QoS and to minimize the application complexity. The aim of each embedded developer is to resolve this equation's system.

One of the used solutions for the previous problem is the adaptable systems: this technique helps to have the best QoS when needed and decrease it we don't. We can use also the hardware accelerators to minimize the software use and to have an important earn in time's execution and consequently a better embedded applications quality of service.

In this paper, we present preliminary results for the design of an adaptive embedded system based on reconfigurable HW variable units dedicated to augmented reality applications.

The paper is organized as follow. Section II gives our work major features and compares it with related works. Section III presents the augmented reality application and talks about proposed architecture. Section IV talks about the Partial reconfiguration and estimated hardware IP results. Finally, section VI concludes the paper with a brief outlook on future works.

II. STATE OF ART

The state of art chapter contains 2 important parts which are adaptive systems and augmented reality. We'll introduce our demonstrator.

1. Adaptatif systems:

Various adaptation techniques have been proposed in to ameliorate the QOS. These techniques touch different levels: application, operating system or hardware levels. Due to the complexity of embedded system, their ability to support new services, the limited energy and the need for mobility, a new type of adaptation techniques is required. It is necessary to adopt global adaptation strategy which combines the previously described adaptation methods.

Several techniques in the domain of self-adaptation have been tested to embedded systems. These adaptation techniques can be applied to the following layers: architecture, operating system and application. We describe the advantages of the proposed techniques as well as their limitations.

a. Architecture (HW)-level adaptation

HW adaptation has been applied to reconfigurable platforms. The change of the system architecture is made according to both the needs of the application and the system's constraints. Such a method is applied to partition the system using two techniques, the first of which uses a heuristic algorithm [1] while the second uses a genetic algorithm [2].

b. OS- level adaptation

OS and middleware layers to providing predictable CPU allocation and adaptation services of OS have been treated [3]. In some researches, the managers of CPU resources provide performance guarantees in “soft” real time. Adaptation is based on the dynamic change of the scheduling policy in [3] to handle the variations of application runtime.

c. Application-level Adaptation

In many projects, adaptation is used at the application layer for different purposes. The authors explore the technique of adapting the behavior of the application to the constraints of energy consumption in [4]. Mesarina et al. [5] discuss how to reduce the energy for the decoding MPEG application using parameter modification. In [6], to improve the constraints of resources and network bandwidth, two approaches are proposed for the deterioration of the quality of 3D.

d. Cross layer adaptation

Previous adaptations are not orthogonal; therefore, cross layer adaptation has been proposed to combine their benefits according to interlayer dependencies. Most of the previous researches were based on methods, which work simultaneously on the various layers of the system such as GRACE 1 and 2 [7,8].

Our work, based on Kais Loukil and al [9] research, add the partial reconfiguration as an adaptation technique. He need in his PHD to embed configurations on the FPGA. The application become complex particularly energy consumption. To solve this problem we introduce the partial reconfiguration part. We need with this technique to embed only the used configuration on the FPGA and not the whole. We choose the 3D application to obtain the augmented [10] reality demonstrator.

2. Augmented reality:

AR technique consists to enhance real video sequences with virtual objects. [11] The AR touches many fields such as : medicine (3D organs modeling…), military (Head-Up Display), industrial (total immersion, remote maintenance [12]), marketing and commercial (advertisements, virtual visits…), entertainments (video games and sport events (player numbers, offside virtual lines, WR comparison line, give visual information for TV viewers from hidden angles in sport match [12] …). [14]

Our target AR application (see Figure 1) is the combination of a video flow recorded with a camera and images synthesis.

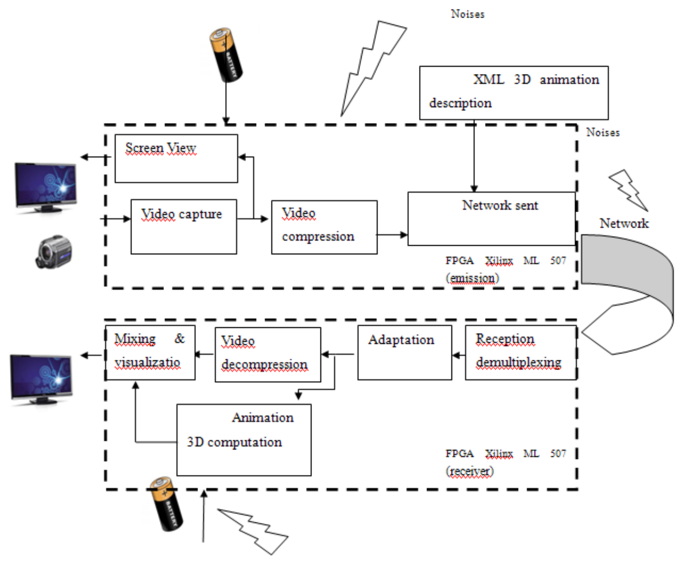

Figure 1: Application demonstrator

Our demonstrator is composed by two parts: a transmitter and a receiver. A camera is used for video acquisition. This camera transmits a video flow to the transmitter. The transmitter is composed of a video flow transformation bloc and a MJPEG coder (embedded in a first ML 507 FPGA board) which is used to compress the video. The 3D animations specifications are multiplexed with the encoded video. They are sent over the TCP IP network using an XML file. At the reception, the video is decoded; 3D animations are computed using XML specifications and mixed with the decoded file before being projected on the screen.

III. Augmented reality and used architectures

The advances in the field of computer vision and mobile computing have made possible the development of complex but one of the main issues remains outdoor application in unknown environment. Applications become more complex, and the environment conditions are unpredictable (sunlight, unrestricted mobility, etc.) and where different types of sensors can be used. [13] In this paper we'll talk about the 3D application implementation.

Adopted architectures:

3D computation requires high performance architecture. GPP are greedy. A solution would be to adopt specific computing units (shader, geometric) these units will be described in the section IV, part B.

To work with complex 3D applications, different GPUs are used.. ATI and NVIDIA leaders of GPUs used different architectures to display 3D images[15].

Inspired from the NVIDIA architecture, we present an architecture based on geometry shaders (GS) for different movements (translation, rotation …) and vertex shaders (VS) (image textures…). The first architectures consist on using one GS and one VS. After that we parallelize the application by adding shader and making some of them working a the same time and the other pipelined.

The number and type of shaders used depends on the type of images and objects. With a richly textured video, many vertex shaders are required with moving video, geometry shaders. GPU use a unified shader to display the video. This architecture is an optimal one for the process using but need many resources.

In the next part, we present an alternative solution based on reconfigurable architecture for 3D objects displaying. Our job is oriented to add the 3D object to the video. This architecture insures a trade-off between the video quality display and the limited FPGA resources.

IV. Application design and hardware accelerator:

A. *3D image synthesis application overview:.*

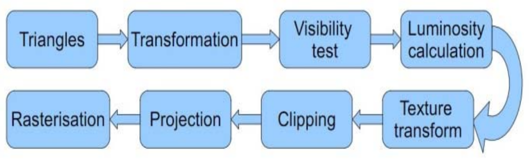

Figure 2: Graphical 3D pipeline

The triangles represent the input of our 3D graphical pipeline (figure 2). The transformation step represents the conversion from local coordinate system to a global one, which is the camera coordinate system. We'll use translations, rotations and homoteties to obtain the final result.

The visibility test consists in identifying which pixel will be viewed and which one will be hidden on the screen using the angle between the vision vector and the hidden one.

The luminosity calculation step gives the luminous intensity attributed to each pixel.

The clipping step consists in eliminating the pixel which will not be on the projected screen but on the computer monitor: if the pixel is a hidden one, it is not displayed.

The projection step is the application of the projective geometry which consists on how displaying a 3D point on a 2D scene.

The rasterisation step is very important because it gives the projected 2D objet a 3D visual aspect when it is projected on the screen. Because of the complexity of the 3D application, we accelerate this software application by introducing hardware blocks. Inspired by the GPP architecture, the software bloc communicates with hardware blocks with the FSL bus.

B. Application analysis and profiling

We use a 3D application available as a C code. In this application the object rotates around different axis. Due to its complexity, the software application version can be displayed but are so slow.

To know which functions must be transformed on a hardware block, we analyze the functions and particularly arithmetic used operations that consume the major part of execution time.

The 3D application study divides the function in two important parts:

- Geometric shader
- Vertex shader

We'll describe in the next the two shaders.

C. Geometric shader

The geometric shader represents the 3D application geometric functions. The figure 3 describes the geometric shader.

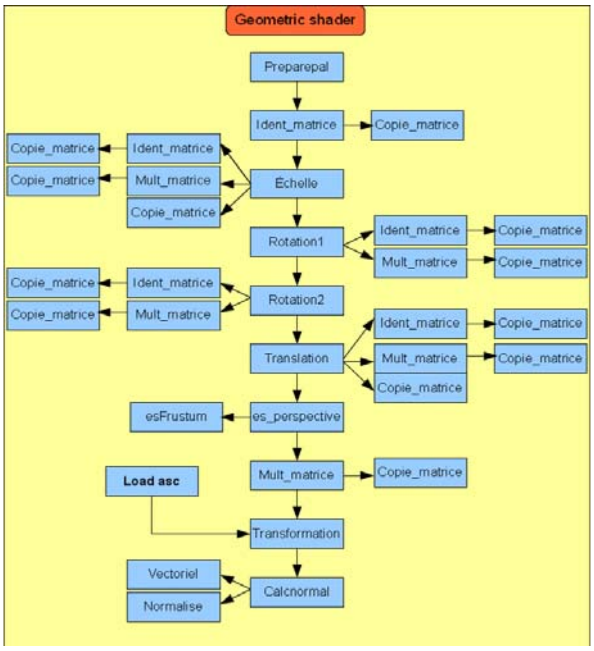

Figure 3: 3D application geometric shader

The geometric pipeline is composed by two parts:

a) Preparation functions:

- The used functions to prepare the geometric object move matrixare: Preparpal : permits the color level calculation.
- Identity matrix: create an identity matrix
- Scale matrix: zoom the object to obtain the needed object scale.
- Rotation: calculates the rotation of the used object.
- Translation: calculates the translation of the used object.
- Perspective transform : calculates the object coordinates projective geometric.
- Matrix multiplication: calculates the multiplication of two 4*4 matrix.

The 3D's C code is profiled to obtain the number of allocated cycles of each function.

Table 1: 3D function profiling

Function	Allocated cycles	Occurrences	% of an occurence
Barycentre	194244	3* nb of objects	4,40%
Normalize	2397997	3* nb of objects	54,39%
Transformation	325676	3* nb of objects	7,38%
Mult_mytice	22640	1	0,51%
Esperspective	22688	1	0,51%
Translation	35330	1	0,80%
Rotation	45774	2	1,03%
Echelle	32262	1	0,73%
Ident_matrice	146	1	0,003%
Ident_matrice	403	1	0,01%
Loadasm	922750	1	20,93%
	4408764		1

The application's profiling result is described in table 1. The calcnormal uses 54% of the code time. This function precedes loadasm the transformation and the Barycentre one.

Loadasm is used to load the polygon coordinates. That's why we don't describe it on the 3D application functions pipelines. The other functions are repeated. We describe the repated functions in the next paragraph.

b) Repeated functions:

- Matrix transform: multiply each triangle summit with the generic matrix obtained after the previous described geometric operations.
- Normalization : permits to obtain the normalized surface vector. This function contains two important functions which are vectorial and normalize.
  - ✓ The vectorial function permits to calculate the vectorial product between two vectors.
  - ✓ The normalize function contains many arithmetic operators such as square root, divisions, additions and subtractions.

The matrix transform and normalization functions are computed on every polygons summits of the 3D objet. Each function is used 3 n polygons with n is the number of our object polygons.

D. *Vertex shader:*

The Vertex shader permits to compute each pixel color value to save it on on memory and to make it ready to the VGA controller. The figure 4 describes the Vertex shader functions used to prepare each pixel color value.

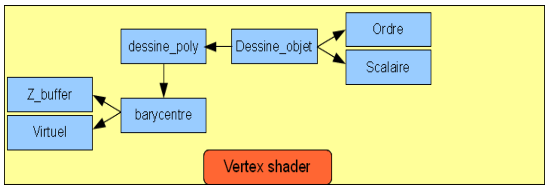

Figure 4 : Vertex shader functions

The vertex shader functions based on the open GL ES principle contains dessine_object function. This function includes dessin_poly function which permits to draw the each polygon. Each pixel color is obtained after that by calculating the barycentre function. This function is the base of Open GL ES 3D application implementation.

V. PARTIAL RECONFIGURATION ARCHITECTURE AND ESTIMATED RESULTS:

In this section, we describe the proposed reconfigurable architecture and the estimated results.

A. *Proposed architecture:*

The adopted reconfigurable architecture is described in the figure 5. There are 3 hardware accelerator zones.

The zone 1 (Z1) contains Normal #1 and Transform #1 as described in Section III, C and b). These functions represented the geometric shader. This zone is a permanent one.. This zone is attached to the microblaze [16] via FSL [16]. The data will be sent to the zone 2 or zone 3 for vertex shading. The microblaze is the Xilinx softcore.

The zone 2 (2) is the reconfigured one. It can be used for not only geometric shader but also a vertex one. If we have a geometrically rich application, the zone 2 is similar to zone 1. In this last case, the FIFO is virtual. We don’t need it in geometric shader but we used it because of the reconfigurable zone. This zone can be also a vertex pipeline. This pipeline needs a FIFO to save the data on it. The Reconfigurable zone needs to have the same input and output despite the configuration type. That’s why the FIFO is always used in the second zone. This zone is also connected to the microblaze via FSL. If the Z2 is similar to Z1 (moved object), the FSL send data to microblaze to be treated after that by the zone 3 (Z3). If the Z2 is similar to Z3 (textured object), the output are saved on FIFO.

The third and final zone contains three blocks. The barycenter block. This block determinates the input of the Dessine-Poly (DP) block. These data are saved on FIFO to be ready for the DP treatment. The Z3 or/and Z2 results are saved after treatment on a FIFO. The two blocks can be used in parallel. For this reason we use the FIFO to save data.

The µB*, is used to transfer data from FIFO to VGA IP via the PLB bus [16]. We can use also a picoblaze to do this task.

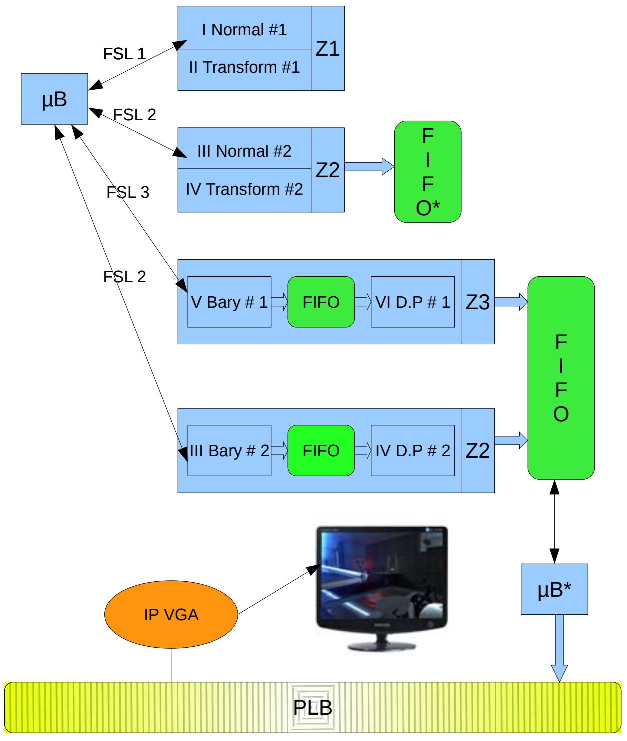

Figure 5: Reconfigurable proposed architecture

B. *Results:*

In this part, we’ll describe the profiling synthesis obtained results and the estimated results of the future implementation jobs.

a. Obtained results:

We describe in this part the results obtained by the hardware simulated block. We use the Xilinx profiling tools to have the aimed results. We compare a software function with a mixed software/hardware (SW/HW) using the Xilinx IP’s core and finally a mixed (SW/HW) using a manually created HW IP. We choose to estimate the normalize function which is one of the most complex one in term of number of arithmetic operators. The table 2 describes the obtained results after the function profiling.

Table2: 3D normalize profiling

Chosen architecture	Number of slices	Execution time
Software	2229	1.2
HW/SW (Xilinx IP)	3656	1.1
HW/SW (ManualIP)	2800	0.9

The software function needs a number of slices smaller than the HW/SW one but need more execution time. We have a gain of 25% of execution time in this case. We can increase this gain and in the same time decrease the number of slices if we use a microblaze without the Floating Point Unit. The FPU unit permits to use arithmetic operators such as division, square roots, sinus, cosines and other arithmetic operators. To improve results, we can transform all the functions in VHDL blocks. With this transformation, we don't need to use the FPU unit and by consequent we gain 1000 slices. (10% of to the FPGA total slice)

We can note that the manual realized VHDL blocks are more efficient than the Xilinx one because their architecture is specified to the application and not a generic one such as Xilinx IP.

b. Estimated results:

Using an only one arithmetic accelerator we obtained a gain of 25% in time execution. We can add to our architecture for the square root and multiplications as hardware blocks in the normalize blocks. We win more than 60% of execution time. In addition, as mentioned in previous paragraph we don't need the use of FPU and consequently we aim many slices.

Approximating these results and using them with the rest of hardware blocks, we obtain the results described in Table 3.

Table3: Accelerator blocks estimation time

Used function	**Execution time cycles**	**Percentage gain**	**Estimation time cycles**
Normalize	2397997	70%	719399
Transformation	325676	40%	195406
Barycentre and Dessine_poly	194244	70 %	58273
Total	2917917	66%	973078

The table 3 that describes the accelerator blocks estimation time proves the important benefits obtained by used accelerator hardware blocks. This gain is obtained by using an only one hardware accelerator for Z1 and Z2 blocks.

The total gain estimation obtained by using hardware block is a 60% of execution time. The same executed code is 0.4 times faster.

VI. Conclusion / Perspectives:

We have presented our concept of flexible architecture for dynamically partial reconfigurable application and detailed the accelerators dedicated to the application. The obtained simulation results are very interesting and make clearer the steps to do for obtaining and augmented reality demonstrator for the 3D application. In the first time, we'll adopt this architectural model and implement it on the Xilinx ML 507 embedded kit. After that, we'll test this approach on different bench marks and adopt it for different 3D video application. According to the complexity of the application in term of texture or movement, we'll adopt automatically the best architectural model.

References

[1] P. Ngoc, G. Lafruit, J-Y. Mignolet , G. Deconinck, and R. Lauwereins "QOS aware HW/SW partitioning on run-time reconfigurable multimedia platforms" Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04, June 21-24, 2004, Las Vegas, Nevada, USA. CSREA Press 2004, ISBN 1-932415-42-4

[2] W. Van Raemdonck, G. Lafruit, E.F.M. Steffens, C.M. Otero Pérez, R.J. Bril "Scalable graphics processing in consumer terminals" Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference

[3] S. Banachowski and S. Brandt, "The BEST scheduler for integrated processing of best-effort and soft real-time processes," in Proc. of SPIE

Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2002.

[4] J. Flinn and M. Satyanarayanan, "PowerScope: A tool for profiling the energy usage of mobile applications," in Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications, Feb. 1999.

[5] M. Mesarina and Y. Turner, "Reduced energy decoding of MPEG streams," in Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2002.

[6] P. Ngoc, G. Lafruit, J-Y. Mignolet , G. Deconinck, and R. Lauwereins "QOS aware HW/SW partitioning on run-time reconfigurable multimedia platforms" Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04, June 21-24, 2004, Las Vegas, Nevada, USA. CSREA Press 2004, ISBN 1-932415-42-4

[7] W. Yuana, K. Nahrstedta, S. V. Advea, D. L. Jonesb, R. H. Kravets "Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems" Appears in SPIE/ACM Multimedia Computing and Networking Conference (MMCN), 2003

[8] GRACE-2: V. Vardhan, D. G. Sachs, W. Yuan, A. F. Harris, S. V. Adve, D. L. Jones, R. H. Kravets, and K. Nahrstedt, "Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy," Int. J.Embedded Systems, 2007

[9] Kais Loukil, "Approche de gestion de performances/contraintes pour les systèmes embarqués temps réel"

[10] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips "Graphics Processing Units, powerful, programmable, and highly parallel are increasingly targeting general-purpose computing applications". GPU-Survey_Proceeding of the IEEE, 2008

[11] Cemil Azizoglu, Ph. D, "High Performance Graphics on Android", Khronos group, 2010

[12] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, M. Morris, and W. Piekarski. "ARQuake :an outdoor/indoor augmented reality first person application". In The Fourth International Symposium on Wearable Computers, 2000.

[13] Gerhard Reitmayr, Tom W. Drummond " Going out: Robust Model-based Tracking for Outdoor Augmented Reality".

[14] Sturman, D.J. and Zeltzer, D., A survey of glove-based input, Computer Graphics and Applications, IEEE , V14 #1, Jan. 1994,30 -39

[15] D.Luebke, SIGGRAPH 2008,Beyond programmable shading in action"GPU Architercutre : Implications & Trends". NVIDIA Coorporation 2007.

[16] www.xilinx.com.