10th International Workshop on Intelligent Solutions in Embedded Systems, 2012

Toward the design of a low cost reconfigurable real time Augmented reality system

Fatma Abbes, Sonda Chtourou, Emna Baklouti

CES Lab, ENIS, National Engineering School of Sfax, Sfax University, Tunisia

Abstract— **In this paper, we detail different step to partially design a low cost reconfigurable real time 3D graphic processor based on the soft core NIOS II CPU. Two major steps are presented in this paper. The first one is the design of a RTOS based on µClinux as OS and Xenomai as real time kernel. We also present already done steps to design a 3D graphic processor. The presented step is the analysis of the 3D graphic application to choose the most complex function for a hardware implementation.**

I. Introduction

An augmented reality (AR) system generates a composite view for the user that is the combination of the real scene viewed by the user and a virtual scene generated by the computer that augments the scene with additional information. [9]. This may be achieved by supplying the user with a head-mounted or handheld display that shows a digitally altered view. Ideally, the virtual objects should merge seamlessly with their real counterparts. Today AR is used many domains such as in entertainment, military training [10], engineering design [12], robotics, manufacturing and other industries.

The great percentage of actual AR systems is based on general purpose processors that execute the processing tasks (even complex) in software. However, software execution is not always the adequate solution especially for the high intensive requirements of the many processing tasks involved in AR. Software only approach inevitably limits frame rate and latency which compromises real time operation. They magnify size and power consumption, reducing the system mobility and autonomy. These limitations make the spread of AR applications more difficult especially in mobile systems.

The rest of the paper is structured as follows. In section II, we present the related work, In section III, we present the adopted method for RTOS integration. In section IV, we detail the design of the 3D synthesis application using the RTOS services and the obtained timing performance. Section V shows preliminary optimization results of the initial system prototype.

II. Related work

Recently, several platforms have been proposed to support mobile AR in two different directions: the head-mounted display direction using PC in backpack systems, and the mobile direction using portable devices. Both directions share a common feature: most AR applications consist of software running on general purpose processors. In spite of their importance, only few of papers describe limited-performance mobile AR systems. In [13], the ArcheoGuide (Augmented Reality-based Cultural Heritage On-site GUIDE) project is presented. It aims at providing visitors of cultural sites with archaeological information. The presented platform has limited performance and causes that the authors eliminate a gesture recognizer due to its interference with the video tracking performance. In [14], the AR-phone is presented. The system performance relies on the wireless networking. Some parts of the most complex processing tasks are moved in remote server to avoid the mobile system overload. A similar system based on a mobile AR is presented in [15] where the data transmission between the phone and the PC is about few seconds. In [18], an AR on-demand system is presented. It is a useful solution on low-end nomad devices where images of the real scene are taken only when needed, superposing real-time virtual animations on that single still image only. In [16], An FPGA-based platform for the development of AR applications is presented. It is composed of a frame grabber for video conversion, a general purpose user interface for visualization and text information insertion and a pointing device that detects the user hand from incoming camera images. This system can process 640×480 pixel images at more than 190 frames per second with a latency of one frame. However, this system is not equipped with an OS and hard to integrate into larger system as it is too specific. In [11], an embedded AR system is presented. It includes an integrated camera and built in graphics acceleration hardware. This system is built around the Overocam camera board that uses an OMAP 3530. The presented application is the tracking of a single marker and rendering a simple virtual object. It runs at around 8 frames per second. Like [16] system, RTOS integration issues are not discussed.

Several AR systems have to be embedded in larger systems such as in robotic, avionic, medicine and military domains. Such complex systems require the use of an OS that can easily manage the complexity at both architectural and application. Several of those systems require real time execution. An RTOS provides responsiveness, determinism and offers valuable services. But in return, it consumes consistent CPU cycles, thereby imposing a processing overhead. So, many issues must be resolved for a successful RTOS integration in AR complex real time systems.

III. RTOS INTEGRATION METHOD

A. *System overview*

The work described in this paper is part of the Cesame project [17]. It aims to build a highly adaptive multimedia system for a distributed AR application on two embedded FPGA-based systems. The considered AR application consists in the addition of 3D effects on specific video objects (identified using segmentation technique). The proposed system can manage various antagonist constraints such as power consumption, available bandwidth, real time constraints and delivered quality. In this paper, we concentrate on the 3D graphics synthesis part of the AR application. We describe firstly the different steps used to implement an RTOS based on Embedded Linux for FPGA platform. Then, we present different performance result of the configured RTOS. Then we describe the 3D application set up and its integration into the built RTOS.

B. *Toward Real Time linux*

We adopt a dual kernel approach which involves two major phases. First, we implement an Embedded Linux kernel (µClinux) for a FPGA prototype platform. Then, we integrated Xenomai, a hard real-time micro-kernel, into Linux kernel in order to enhance it with hard real-time behavior. Real-time schedulers select the ready task which has the highest priority. The kernel must be deterministic and preemptive. Standard Linux is not an RTOS because it has long sections of code [3]. Linux kernel is not completely preemptive. Therefore, a process will be interrupted only if another process is running in user space and does not require a kernel routine.

Several technical solutions have been proposed to enhance the behavior of the Linux kernel with real-time capabilities. The different solutions are split into two families [4]. The first one is the application of specific patches. The purpose of this solution is to improve the latency of system calls and the reactivity of the scheduler by adding a preemptive level to kernel. These changes don't transform the Linux kernel to a hard real-time kernel, but satisfy soft real time constraints (improvement in the quality of service of multimedia application for example). The second solution is the addition of an auxiliary real time micro-kernel. The defenders of this approach believe that Linux will never be truly real time and therefore add to the kernel another micro-kernel with a true real-time schedule. This approach is adopted by RTLinux, RTAI and Xenomai [5]. In this new "dual core" OS, all interrupts are passed to Linux only if there is a non real time task to execute. This extension provides a deterministic behavior in an environment where the latency is extremely low. The diagram in Fig. 1 describes the architecture of "dual-core" OS.

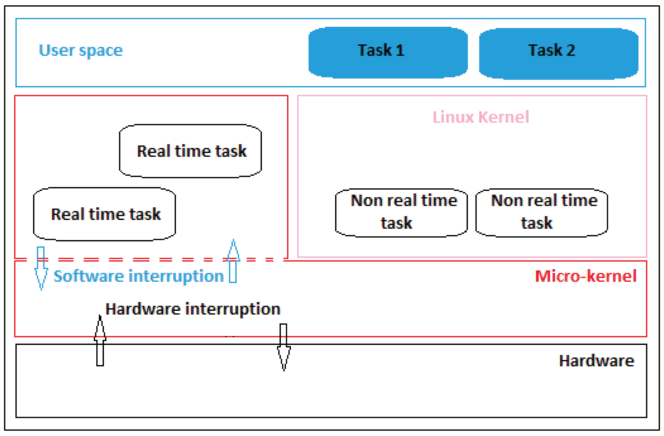


Figure 1. Micro-kernel method

There are various free and commercial micro-kernels. We focus on free distributions like RTAI, Xenomai and primarily version of RTLinux. With RTAI and RTLinux, real-time tasks are performed in kernel space (kernel-space) which can cause many risks and debugging problems. Meanwhile, Xenomai allows the user to develop and implement real-time tasks in user space. In addition, it keeps coherence of priorities between space and Xenomai real time Linux kernel. Moreover, it is possible to make a transparent migration of tasks between these two spaces [3]. Xenomai uses various API like native POSIX, RTAI, pSOS and VxWorks. For all this reasons, we adopt Xenomai as the used real time micro-kernel. We use as OS kernel the µClinux and as target platform the ALTERA Cyclone III FPGA starter kit platform. We adopt a single core CPU architecture based on the NIOS II processor. Two timers are added to the standard version of the NIOS using the Quartus software. The first one is a 32-bit timer used by µClinux. The second one is a high precision 64-bit timer named used by Xenomai.

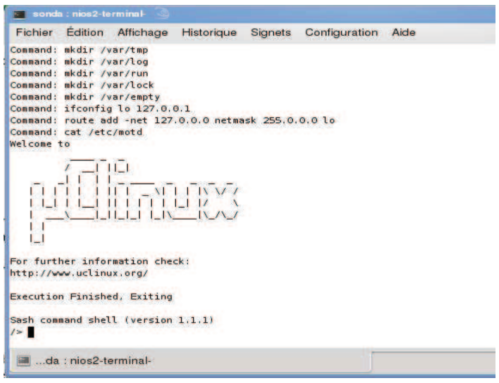
C. *Configuration of µClinux kernel*

After configuring the required hardware design to boot the micro-kernel, we configured the µClinux kernel for Cyclone III platform using a header file provided by the source of µCLinux. This file describes the system addressing space of all hardware components (address of the processor, memory and input / output ports). Finally, we cross compiled kernel for the NIOS processor. The result of cross-compiling is the creation of the binary zImage which represents the executable we will load on the platform.

D. *Integration of micro-kernel Xenomai*

Xenomai has its own environment called "user-space" to be compiled for our embedded system. We configured the user-space Xenomai for our NIOSII processor. Then, we cross-compiled it in order to generate the executables needed for the functioning of Xenomai. We installed the entire generated executables in specific place in the root file system of µClinux kernel. We configure the FPGA platform with the adequate hardware design based using the Quartus compilation process.

Then, we loaded the final binary of the built RTOS (zImage) in the flash memory of the target platform. We executed binary in order to boot the RTOS. As result, the two kernels (µClinux and Xenomai) started with success as shown in Fig. 2.

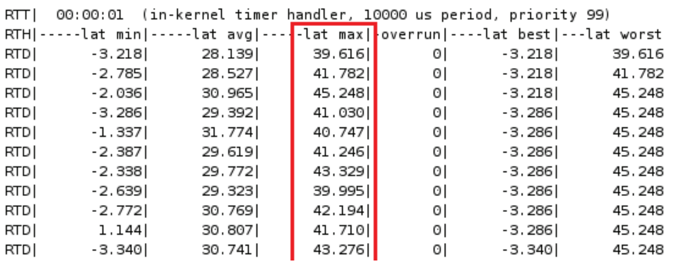


```
sonda : nios2-terminal-
Fichier  Édition  Affichage  Historique  Signets  Configuration  Aide
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo
Command: cat /etc/motd
Welcome to

For further information check:
http://www.uclinux.org/

Execution Finished, Exiting

Sash command shell (version 1.1.1)
/>
...da : nios2-terminal-
```

Figure 2. real time µclinx strat up.

E. *Latency Estimation*

We measured the latency of the added micro-kernel Xenomai. According to previous experiments done on x86 designs, the average latency of Xenomai is 7 µs and the maximum latency time is between 10 and 15 µs [4]. In our case, the average latency is 42 µs as shown in Fig. 3. It is a predictable result as NIOS II has average computing performance of only about 90 MIPS.



```
RTT|  00:00:01  (in-kernel timer handler, 10000 us period, priority 99)
RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst
RTD|      -3.218|      28.139|      39.616|       0|      -3.218|      39.616
RTD|      -2.785|      28.527|      41.782|       0|      -3.218|      41.782
RTD|      -2.036|      30.965|      45.248|       0|      -3.218|      45.248
RTD|      -3.286|      29.392|      41.030|       0|      -3.286|      45.248
RTD|      -1.337|      31.774|      40.747|       0|      -3.286|      45.248
RTD|      -2.387|      29.619|      41.246|       0|      -3.286|      45.248
RTD|      -2.338|      29.772|      43.329|       0|      -3.286|      45.248
RTD|      -2.639|      29.323|      39.995|       0|      -3.286|      45.248
RTD|      -2.772|      30.769|      42.194|       0|      -3.286|      45.248
RTD|       1.144|      30.807|      41.710|       0|      -3.286|      45.248
RTD|      -3.340|      30.741|      43.276|       0|      -3.340|      45.248
```

Figure 3. Latency of Xenomai

IV. INTEGRATION OF THE RTOS SERVICES WITH THE 3D IMAGE SYNTHESIS APPLICATION

Once our RTOS is successfully loaded on the FPGA platform, we aim to implement a 3D image synthesis application using its different services. We first run the 3D application as one task executed on our new RTOS µClinux.

A. *3D image synthesis overview*

The 3D image synthesis is described in Fig 4. Our application is conform to the OpenGL ES 1.X specification [6]. It takes as entry point an existing 3D model of the object to render and generate various 3D animations. The 3D object is modeled with a set of triangles stored in a text file (.asc or .vrl). Transformation step handle the animation of the 3D object (zoom, motion, etc). Clipping limits computations on only the visible part of the 3D object on the screen. Perspective division performed on the Clip Coordinates produces normalized device coordinates. Illumination step computes the different triangles colors. Texture mapping consist on applying a 2D map to the surface of a polygon. The viewport transformation maps normalized device coordinates into window (screen) coordinates. The rasterization step then decides which pixels of the frame buffer the primitives cover, and which colors and depth values those pixels are assigned. Z-buffer algorithm is used to eliminate hidden pixels.

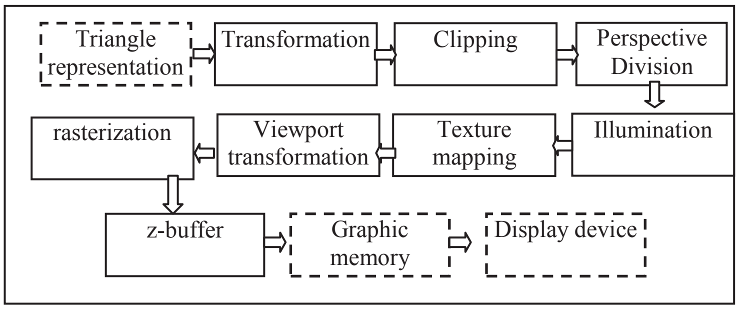
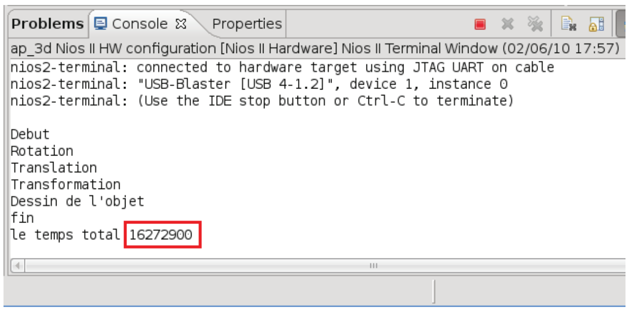


Figure 4. Graphical 3D pipeline

B. *3D application Execution under NIOS processor*

The 3D application was first tested on PC platform. It was then adapted and tested on the NIOS II processor using ALTERA Cyclone III FPGA starter kit platform. This implementation is purely software and used to verify the correctness of the NIOS-adapted application. As the used platform is devoid of a VGA port, we compare the 3D output application (the color matrix generated after the z-buffer algorithm) of both NIOS adapted version and the PC version. Fig. 5 shows the timing results of the application (without µClinux). Execution time is about 0.16s

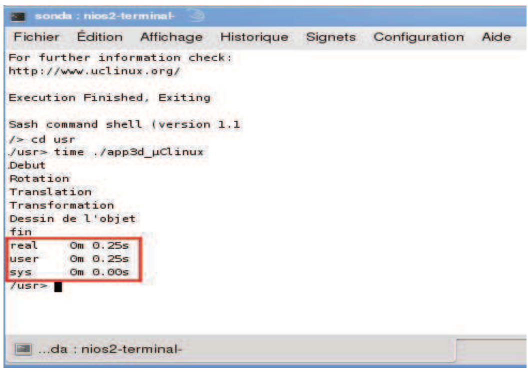


```
ap_3d Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (02/06/10 17:57)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB 4-1.2]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Debut
Rotation
Translation
Transformation
Dessin de l'objet
fin
le temps total 16272900
```

Figure 5. Execution of 3D application on NIOS CPU.

C. *3D application Execution under µClinux*

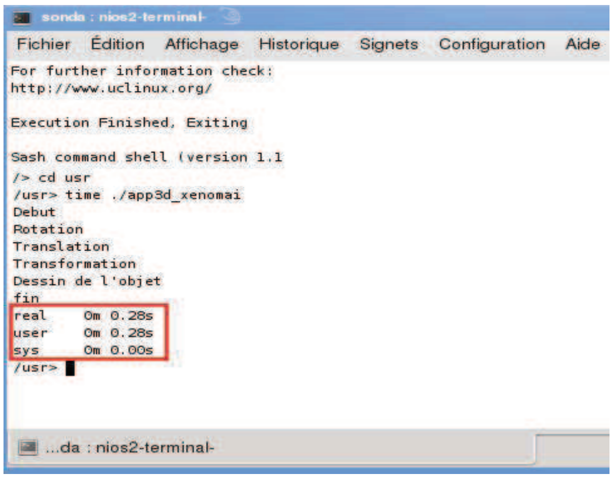
We run the 3D application as a non real time application (one task) on the built Real time µClinux. Figure 6 shows the associated execution time (0.25s). The extra time (0.25-0.16=0.09s) is due to the µClinux overhead.


sonda : nios2-terminal-
Fichier Édition Affichage Historique Signets Configuration Aide
...da : nios2-terminal-

```
For further information check:
http://www.uclinux.org/

Execution Finished, Exiting

Sash command shell (version 1.1
/> cd usr
/usr> time ./app3d_µClinux
Debut
Rotation
Translation
Transformation
Dessin de l'objet
fin
real     0m 0.25s
user     0m 0.25s
sys      0m 0.00s
/usr>
```

Figure 6. 3D application execution on µClinux

We run the 3D application as a real time application (executed on Xenomai) as shown in figure 7. Execution time is about 0.28s. The additional 0.03s is due to the high preemption of the Xenomai kernel compared to the µClinux kernel.


sonda : nios2-terminal-
Fichier Édition Affichage Historique Signets Configuration Aide
...da : nios2-terminal-

```
For further information check:
http://www.uclinux.org/

Execution Finished, Exiting

Sash command shell (version 1.1
/> cd usr
/usr> time ./app3d_xenomai
Debut
Rotation
Translation
Transformation
Dessin de l'objet
fin
real     0m 0.28s
user     0m 0.28s
sys      0m 0.00s
/usr>
```

Figure 7. 3D application execution on Xenomai

D. *RTOS service integration*

In this section, we analyze the 3D application execution using the RTOS different services. In this case, the application is split into several tasks that communicate using the different RTOS services. We take as 3D application scenario the simultaneous rendering of 3 different objects with different resolutions (triangle number) and various animations. We create five communicating tasks. Fig. 8 shows the application scenario. The first startup task "TaskStart" creates simultaneously four tasks "Task_Anim_1", "Task_Anim_2", and "Task_Anim_3" and "Task_Mixage". Each "Task_Anim_i" creates and prepares the final pixel color matrix associated to one iteration the animation a 3D object "i". "Task_Mixage" assembles and render the three objects in a single screen. "Task_Mixage" adds the three generated pixels color matrix to obtain the final 3D scene to render. "Task_Mixage" has to wait the end of execution of three animation tasks. This wait state is realized through a "semaphore" service. "Task_Anim_1" and "Task_Anim_2" has to render a duplicate 3D objet with different animations. They use the same 3D object representation stored in memory. This shared resource (i.e the memory object representation) is protected with a Mutex service. In our scenario, we also imposed that "Task_Anim_3" waits the end of execution of "Task_Anim_1" using a semaphore service. "Task_Anim1" uses the services of message queue to send a message to "Task_Anim2".

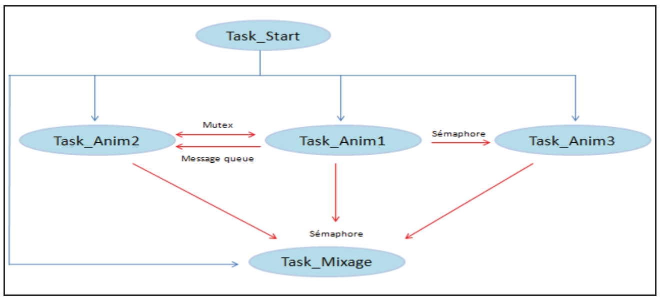


Figure 8. Split the 3D application into tasks

E. *Implementation of the scenario*

To execute the application scenario on µClinux, we use POSIX API to create non real time threads. This application will be executed as a non real time application. To execute the implemented scenario with Xenomai, we used native API to create non real time threads. Table 1 shows the obtained results after execution on the FPGA kit.

TABLE I. OVERVIEW VALUES

Service	Xenomai	µClinux
Semaphore	2814 µs	545 µs
Mutex: Acquisition	936 µs	48 µs
Mutex : liberation	930 µs	276 µs
Message_queue	4076 µs	4882 µs

We remark that the overheads of the various services of the micro-kernel Xenomai are higher compared to the µClinux. It can be explained by the high number of call the Xenomai scheduler. This hard real-time scheduler has to always verify if there is a new ready task that has a higher priority than the current task.

V. SYSTEM OPTIMIZATION

The performance of the designed system is limited. They are related essentially of the poor performance of the NIOS II CPU and the complexity of the 3D application. In this section, we propose several techniques to enhance system's performance.

A. Application optimization

It is possible to reduce the CPU workload using application parameter or algorithm tuning without noticeable quality degradation. In the case of the 3D application, its execution time is proportional to the 3D object resolution [7]. An automatic resolution reduction is applied when the 3D object became too small (when a zoom out is applied). The reverse operation occurs when a zoom out is applied to 3D objet.

B. Architecture optimization opportunities

As we use FPGA technology in our system, an efficient architecture can be designed to the 3D synthesis application even with based on a limited CPU such as the NIOS II processor. To develop high performance architecture for the 3D application, it is necessary to identify both the often-called functions (OF functions) and the high complexity functions (HC functions). Both (OF and HC) functions implementation (hardware or software) type influence the overall system performance. By choosing suitable (hardware) implementation of (OF and HC) functions, system performance can be enhanced sharply.

We analyze the 3D synthesis application using profiling technique. In this stage, we use a 3D application version without texture mapping step. As we plan the use of the NIOS embedded processor, we adopt its associated profiling tool "NIOS IDE". It has a "Performance Counter" module that determines the different functions execution time, their percentage as well as the call number of each function.

Figure 9 shows the organization of the different functions of the 3D application.

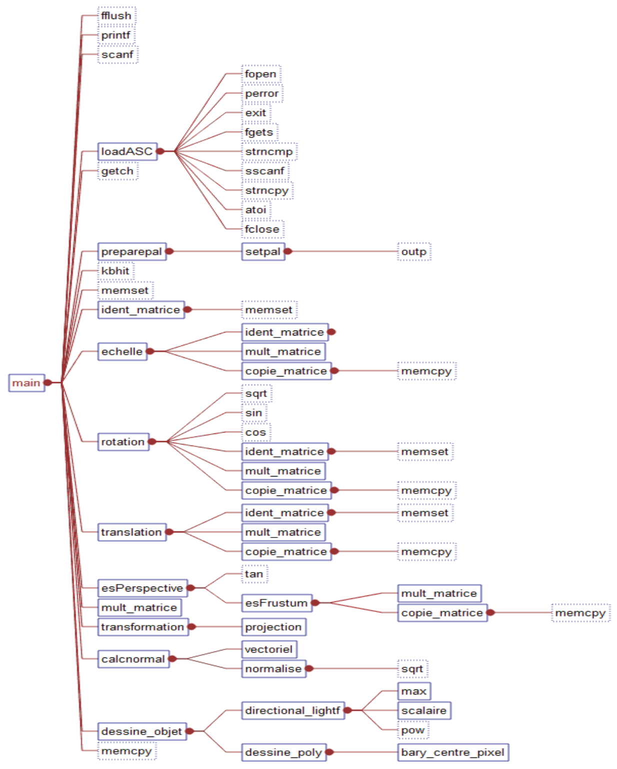


Figure 9. 3D Application functions organisation

Profiling results for a single computing iteration is shown in table II. Dessine_object function (that corresponds to the rasterization function in fig. 4) monopolizes 97% of the total execution time.

TABLE II. PROFILING RESULTS

Function	**Pourcentage**
Dessine objet	97,5849%
Calcnormal	1,3741%
Transformation	0,3953%
mult_mytice	0,0130%
Esperspective	0,0130%
translation	0,0202%
rotation	0,0517%
échelle	0,0185%
ident_matrice	0,0003%
loadasm	0,5288%

TABLE III. ANALYSIS OF THE "DESSINE_OBJET" FUNCTION

Function	Cycle number	Call number
Dessine Objet	170294048	1
Dessine Poly	167036485	40
Barycentre	166273113	856

TABLE IV. COMPLEXITY COMPARAISON BETWEEN 3D APPLICATION SUB FUNCTIONS

function	NIOS Cycle number
Calcnormal	2397997
Transformation	999956
Loadasm	922750
Barycentre	194244
Rotation	90348
Translation	35330
echelle	32262
Esperspective	22688
mult_mytice	22640
ident_matrice	403
ident_matrice 1	146

We profile the most complex function "Dessine_objet". Results are in Table II. They are specific for a single iteration. Table 2 shows that the "barycentre" function (corresponds to the rasterization step) is the most called function in the 3D application. We consider it as OF functions. It has to be implemented in hardware. We also compare relative complexity of all sub functions of the 3D application using the profiling technique. Results are in table IV. They are specific for a single computing iteration. We notice that the “Calc_normal” function is the most complex sub_function. This function is called every computing iteration. It uses complex mathematical computations such as “sqr” and “sqrt” operator. We consider it as HC function. It has to be implemented in hardware.

C. Barycentre function analysis:

Barycentre function is part of the rasterization step of the 3D image synthesis. Rasterization is the process by which a triangle is converted to a two-dimensional image. Three step are required: 1 triangle summit normal compute, 2 intensity summit compute, 3 color computing for the different pixels of the triangle using barycentre interpolation. Barycentric coordinates are a set of three numbers α, β, γ, each in the range [0; 1], with α+β+γ=1. The color « c » of a pixel within the triangle (p1, p2, p3) with respective colors (c1, c2, c3) is computed using (1). α,β,γ are computed using an interpolation technique using equation (1) [6].

$$c = \alpha c_1 + \beta c2 + \gamma c3 \quad (1)$$

Color computation depends only on the pixel position (barycentric coordinates). There is no correlation between two successive pixels color like in Gouraud shading algorithm. So, parallel computations can be carried to accelerate this function. Two parallelism levels can be noted in color computation. The first one is “intra_triangle” parallelism. Within the same triangle, several pixels color computation can be carried simultaneously since there no correlation. The second parallelism is “inter_triangle”. Concurrent triangles color computation can also be simultaneous. To fully exploit all these parallelism opportunities, a SIMD accelerator can be envisaged. Design details about this accelerator will be published in future papers.

D. RTOS overhead reduction

An RTOS provides responsiveness, determinism and offers valuable services. But in return, it consumes consistent CPU cycles, thereby imposing a processing overhead as shown in table I. As a consequence, the time available for user tasks is reduced. To reduce we proposed to hardware accelerate an RTOS. The idea is to convert a series of RTOS instructions into a simple instruction implemented in hardware. We plan to use the same technique as in [8].

VI. Conclusion

In this paper, we detail different step to partially design a low cost reconfigurable real time 3D graphic processor based on the soft core NIOS II CPU. Two major steps are presented in this paper. The first one is the design of a RTOS based on μClinux as OS and Xenomai as real time kernel. We also present already done steps to design a 3D graphic processor based on the NIOS II CPU. The presented step is the analysis of the 3D graphic application to choose the most complex function for a hardware implementation.

References

[1] www.ghs.com/news/pdfs/alt-software-and-greenhills.pdf

[2] A. Burns and A.J. Wellings, “Designing Hard Real-time Systems”, Proceedings of the 11th Ada-Europe Conference, Lecture Notes in Computer Science, Vol 603, pp116-127, 1992.

[3] http://www.unixgarden.com/index.php/comprendre/temps-reel-sous-linux-reloaded.

[4] P. Kadionik, "Linux embarqué, Linux Temps Réel : présentation". ENSEIRB, Décembre 2003.

[5] P. Ficheux et P. Kadionik, "Temps réel sous LINUX", online course available at , http://pficheux.free.fr/articles/lmf/realtime/linux_realtime4_img.pdf.

[6] Open GL ES specification url : http://www.khronos.org/opengles/1_X

[7] N. Ben Amor « Conception d’un processeur de vision embarqué », PhD thesis, Sfax University, Tunisia,2005.

[8] I. Feki, “Hardware RTOS Acceleration”, master thesis, National Engineering School of Sfax, 2010.

[9] [AR]: URL : http://www.webopedia.com/TERM/A/Augmented_Reality.html

[10] D.G. Brown,Y. Baillot, S.J. Julier, P. Maassel “ Building a Mobile Augmented Reality System for Embedded Training: Lessons Learned, ” Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2004.

[11] M. Groufsky, “An embedded augmented reality system ,” master thesis, University of Canterbury, Christchurch, New Zealand., February 2011 available at http://ir.canterbury.ac.nz/handle/10092/6259

[12] W.A. Khan, A. Raouf, K. Cheng, Virtual Manufacturing, Springer series in Advanced Manufactoring, Springer 2011.

[13] Dähne, P. & Kariginannis, J. (2002). Archeoguide: system architecture of a mobile outdoor AR system, Proc. of International Symposium on Mixed and Augmented Reality, pp. 263-264, 0-7695-1781-1, Darmstadt, Germany, September 2002, IEEE CS

[14] Assad, M.; Carmichael, D.; Cutting, D. & Hudson, A. (2003). AR phone: Accessible Augmented Reality in the Intelligent Environment, Proc. of Australiasian Computer Human Interaction Conference, pp. 232-235, Queensland, Australia, 2003

[15] M. Billinghurst, M. Hakkarainen & C. Woodward, “Augmented Assembly using a Mobile Phone,” Proc. of International Conference on Mobile and Ubiquitous Multimedia, pp. 84-87, 978-1-60558-192-7, Umea, Sweden, december 2008.

[16] J. Toledo, J. J. Martínez, J. Garrigós, R. Toledo-Moreo and J. M. Ferrández, “Design of Embedded Augmented Reality Systems”, chapter book the horizon of Virtual and Augmented Reality available at http://www.intechopen.com/books/augmented-reality.

[17] Césame project url: http://www.ceslab.org/eng/projet_det.php?id_projet=10

[18] P. Riess, D. Stricker, “AR on-demand: a practicable solution for augmented reality on low-end handheld devices”, Proc. of AR/VR Workshop of the German Computer Science Society, pp. 119-130, 3-8322-5474-9, Coblence, Germany, 2006