A DATA ADAPTATION APPROACH FOR A HW/SW MIXED ARCHITECTURE

(CASE STUDY: 3D APPLICATION)

Tarek FRIKHA, Nader BEN AMOR, Khaled Lahbib, Jean-Philippe DIGUET*, Mohamed ABID
CES-Laboratory,Lab-STICC*
Sfax University, National Engineering School of Sfax, Sfax, TUNISIE
Université de Bretagne SUD, Lorient, France
Sfax TUNISIA

Abstract: This Embeddd systems use emerges in the electronic field. Many applications have been embedded such as network, image processing, signal processing … The emergency of multimedia applications particularly in mobile embedded systems puts new challenges for the design of such systems. The major difficulty is the embedded system's reduced computational resources that must be carefully exploited to execute variable workload due to many parameters such as data. In this paper, we propose an adaptive architecture based on dynamical partially reconfigurable approach that manages the system architecture complexity according to the data variability. The augmented reality application is the case study application to confirm the proposed approach.

Key-Words: - Data adaptation, dynamical partially reconfiguration, augmented reality, 3D application, hardware accelerator, hardware software adequacy.

1 Introduction:

The multimedia embedded applications inflate the computer sciences domain. Watching for example to a HD video or a 3D movie is now possible not only with a 3D TV but also possible on small portable systems such as smartphone and tablets.

The design of such systems faces many challenges due to the limited available resources and the external environment fluctuations such as noise, application variation, and dynamic nature of applications… To increase embedded application quality, we use adaptation techniques.

The adaptation techniques concern many layers such as application, OS, network, middleware, hardware [1]…

In the hardware adaptation layer, the hardware/software (HW/SW) adequacy of systems is a promising solution. The software application use permits to have the easiest solution whereas the mixed one is more complex and increases the quality of service of the application.

In this paper, we propose an approach based on data adaptation. According to the input data, we modify the mixed HW/SW application. As case study, we choose an augmented reality application (AR). The described approach is tested on the 3D application.

AR technique consists to enhance real video sequences with virtual objects. [2] The AR touches many fields such as : medicine (3D organs modeling…), military (Head-Up Display), industrial (total immersion, remote maintenance [3]), 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 [3] …). [4]

Our target AR application is the combination of a video flow recorded with a camera and 3D images synthesis.

The paper is organized as follows. Section II gives our work major features and compares it with related works. Section III presents our application design and details our Hardware accelerator. Section IV shows the implementation and the obtained results. Finally, section V concludes the paper with a brief outlook on future works.

2 State of art:

2.1. Adaptive techniques:

The augmented reality application needs a real time execution. The 3D application insertion must be coherent with the video flow. That's why we

have to adapt the 3D application. Various adaptation techniques have been proposed to ameliorate the quality of application. To adapt the application, many techniques can help us to create an optimized architecture. We can use the Multifacet's general execution-driven multiprocessor simulator (GEM 5) simulator for modelling a multicore embedded system [5]. Many different works used adaptive techniques. The adaptive techniques touch monoprocessor and multiprocessor architectures. We will expose in this state of art some related works done for each type of architectures.

2.1.1 Monoprocessor architectures:

Many works used monoprocessor architectures for different applications. For 3D applications Kais Loukil & all used monoprocessor architecture for hardware application accelerators. The created accelerators used Altera IP's. [1]
Kais Loukil used an adaptation approach based on application, OS, middleware and hardware layers. He proposed an approach validated by a 3D application but don't depend on the data adaptation. The used architecture does not change if the coordinates number is modified in the application input.

2.1.2 Multiprocessor architectures:

Many applications used multiprocessor architectures for different multimedia applications. L.F Ye and all, used in [7] multiprocessor architecture with partial reconfiguration application. Y.Corre & all [8] used MPSOC architecture for an MJPEG application. The proposed architecture based on Xilinx platform used 2 microblazes. It optimized the application complexity with the quality of service (QoS). In [8], the adaptation techniques allow obtaining a decompressed video turning with a better quality.

2.1.3 Approach setup:

In our application, we choose to adopt a monoprocessor architecture using a microblaze with hardware IP described in VHDL language. The augmented reality adaptation concerns the 3D application. We will use the adaptation technique to the application. The figure 1 describes our adaptation technique for the chosen application.

According to the input coordinate number we adapt our architecture. If the number of coordinates increases, the adapted architecture will change. The architecture and particularly the developed IPs depend on the number of data. In the next part, we will describe the application design and the used hardware accelerator.

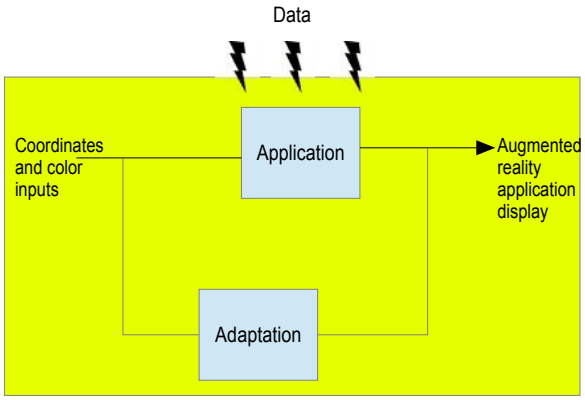

Figure 1: Application adaptation

2.2. Data adaptable computing :

As described previously, our application concerns the data adaptation. In most multimedia applications, the data profile of the incoming data is contained within the header of an incoming data stream, which can be examined to determine if a suitable hardware implementation is available for any of the application's tasks. The combined HW/SW implementation uses different accelerators. According to the application data input, we choose and implement the adequate accelerator.

Many approaches permits to generate from data inputs a mixed HW/SW reconfigurable application implemented in FPGA platforms. The Data-Adaptable Reconfigurable Embedded System (DARES) approach [9] is based on generating a mixed HW/SW reconfigurable application. DARES also enables the efficient realization of an adaptable HW/SW implementation for many applications such as JPEG, JPEG 2000 [10] and streaming.

For the latter many languages have been developed for streaming application supporting software such as OpenCL, OpenMP [11] and hardware based implementations like ImpulseC. [12]

The DARES approach consists, using an application and Data profile model, to generate a software binary, using software compiler GCC. The hardware tasks are transformed to Hardware IP using ImpulseC CoDeveloper. The hardware task bitstreams are added to the software binary and are implemented on FPGA ML 507.

The hardware generated accelerators are generic. The adaptation part depends on the application change: if we change an application, we create a new architecture. Our approach is based on creating an architecture containing a fixed part and a modifiable one which depend on the data characteristics.

3 Approach validation:

In this part, we'll talk about the 3D application and the chosen approach to validate the embedded application. This classical approach is based on the Gouraud shading algorithm [13].

3.1. Description and implementation of the 3D application:

3.1.1 3D image synthesis application overview:

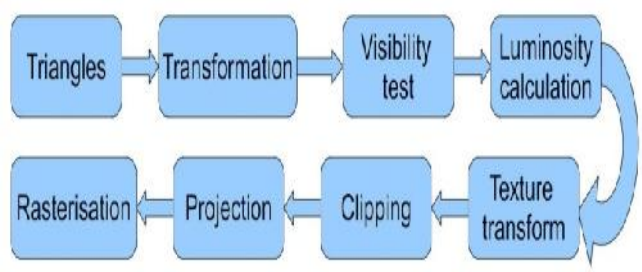

Figure 2: Graphical 3D pipeline

The 3D application is depicted on figure 2. 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.

3.1.2 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 is so slow. Using architecture with a microblaze and without hardware IP, we obtain a 3D application turning into 6.5 seconds.

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 used functions to prepare the geometric object are:

- Triangles:
  - ✓ Preparpal : permits the color level calculation.
- Transformations:
  - ✓ 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.
- Visibility test, Luminosity calculation and Texture transform:
  - ✓ Perspective transform: calculates the object coordinates projective geometric.
  - ✓ Matrix multiplication: calculates the multiplication of two 4*4 matrix.

Some used functions are repeated one. The clipping and projected functions are repeated. These functions are:

- ✓ 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 final group of functions contain the dessine_object function and particularly hline one. This function permits to fill each polygon of the 3D object.

The figure 3 represents the 3D algorithm functions.

We need to accelerate the application by creating hardware blocks replacing the heaviest 3D application functions. To know which functions must be transformed on a hardware block, we profile the native C code via the Xilinx EDK profiling tool.

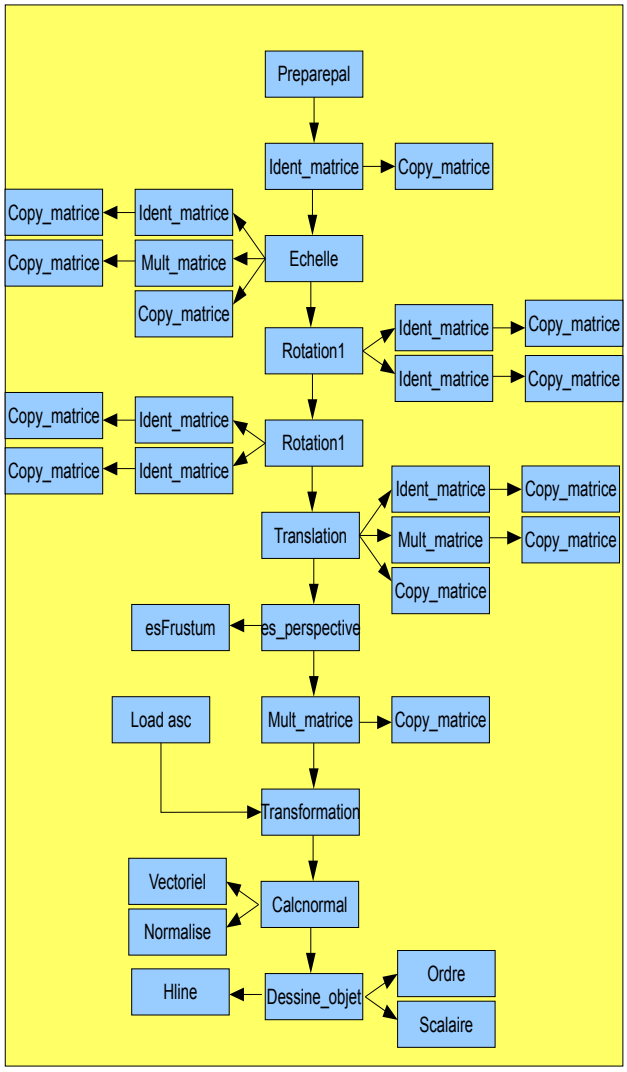

Figure 3: 3D application functions

The table 1 represents the 3D application profiling.

Table 1: 3D's function application profiling

Functions	Time percentage
Hline	69%
Rotation	13%
Echelle	5%
Translation	5%
All other functions	< 2%

The profiling result shows that the polygon filling takes the most important part of 3D application. The 3D application is based on a rotation around an only one axis.

69% of the application time was dedicated to the Hline function which is oriented to fill in each pixel the attributed color value. Each pixel of the triangle contains a value which is an integer that belongs to [0,255]. We must assign the appropriate value to each pixel. However, one may keep in mind that this profiling depends on the benchmark.

All the other application function didn't use less than 13% and that's why we choose the Hline to accelerate it.

The Hline function can be called "pixel shading" whereas the rotation and translations functions represent the geometry's one.

Our 3D object is formed by a set of triangles. Each triangle is filled using horizontal lines formed with pixels. The Hline function attributes to the pixels of each horizontal line of each triangle the color value according to the Gouraud shading algorithm.

3.1.3 Hline hardware accelerator architecture:

The implementation of the hardware accelerator aims to speed up a low frequency low cost architecture to display the 3D object moving on the screen without being heavy.

Our study and implementation is based on Virtex 5 that can be dynamically and partially reconfigured.

Figure 4 represents the accelerator with more details

This accelerator contains two important stages:

- Lines extremities determination
- Pixel color filling.

3.1.2.1 Segment's extremity determination (SED):

The line extremities determination is represented by two steps which are represented in the figure 4:

- ✓ Segment's extremity determination.
- ✓ Segment's pixel extremities filling

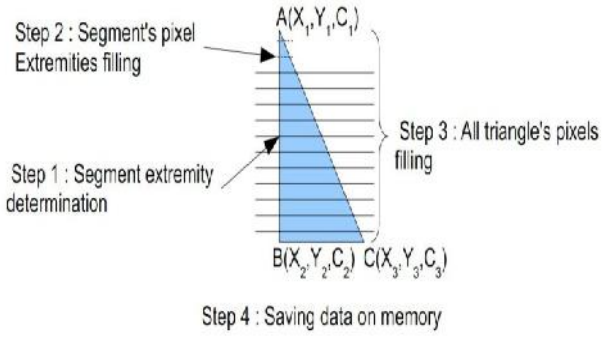

Figure 4: 3D triangle's pixel filling accelerator

Segment extremity determination consists on finding the x coordinate of the pixels which are the extremity of each triangle's line side.

Every triangle is formed by three sides. To fill the entire triangle we have to find the pixels which are on every triangle side. Since the triangle is displayed on the screen our landmark entity will be the pixel not only horizontally but also vertically.

To find the triangle segment's extremity we must find director coefficients of each triangle's side.

Using the three top coordinates, we can calculate the director coefficient using the equations system:

$$y = ax + b \quad (1)$$

$$a = \frac{y_B - y_A}{x_B - x_A} \text{ et } b = y_A - ax_A \quad (2)$$

Each pixel value will be incremented by the director's coefficient ***a***. We skim from the minimum value of the abscise to the maximum one.

To obtain the coefficient a value, a division is required. This arithmetical operation is done by using the IP core generator of Xilinx.

3.1.2.2 Segment's pixel filling(SPF):

Pixel color filling is also based on two steps:

- ✓ All triangles pixels filling.
- ✓ Saving data on memory.

The color value of each pixel obtained is the difference between the vertices colors divided by the difference between Yb and Ya as mentioned in equation (3):

$$c_{inc} = \frac{c_B - c_A}{y_B - y_A} \quad (3)$$

Once all triang les are found, all these values will be saved on the FPGA external dual port BRAM memory [18]. We use a dual port BRAM memory because we need to save the SPF data on the memories. These data will be used as the input of all triangle pixels fillings blocs.

Finally, we will focus on the accelerator which is the Pixel color filling.

3.1.2.3 All triangles pixels filling:

After filling the color value of the triangle's side, we'll find the value of the color of each pixel interior to our triangle. We'll use the same method used for the triangle extremities values.

Then, we'll save the pixels filling values on a BRAM bloc.

3.1.2.4 Saving data on the memory:

The last step of our accelerator will be to save the data obtained on a BRAM memory bloc.

This bloc generated by the Xilinx IP core generator also contains every pixel abscise value and the color that we'll affect to this pixel. These values will be used when we'll obtain the entire image to display on the screen after the entire treatment.

The figure 5 represents the entire 3D accelerator.

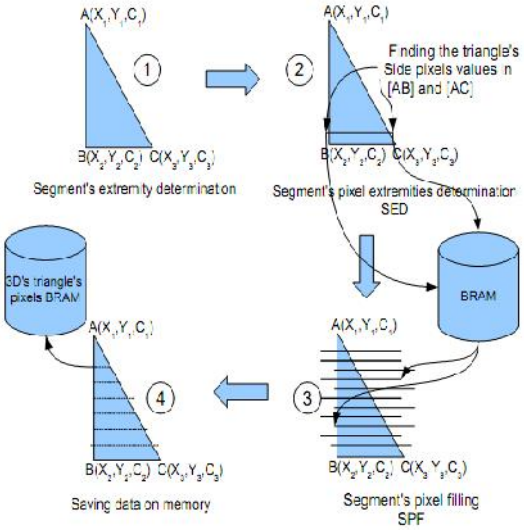

Figure 5: 3D triangle's pixel filling accelerator steps

Global flexible architecture :The whole project lies on a concept of self-adaptive architecture based on a softcore (Microblaze) [13] enhanced with a set of reconfigurable VHDL accelerators (see figure 6). Based on various profiling we can observe that Geometry Computation and Hline accelerators must be considered. However the number of accelerator of each kind can be adapted according to application needs. The Hline accelerator integration is detailed in the next section.

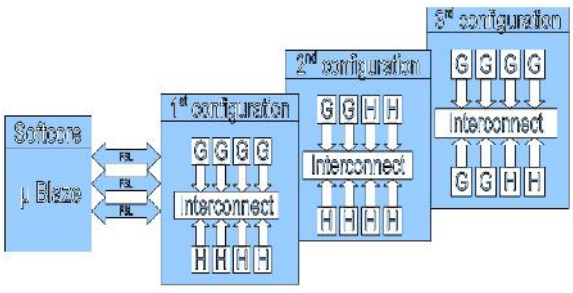

Figure 6: Global architecture: G: Geometry, H: Hline

3.1.2.5 The filling application : Hline architecture

The interface between Microblaze and accelerators is based on FSL bus; since FSL FIFOs provide fast communication between hardware blocks and the processor.

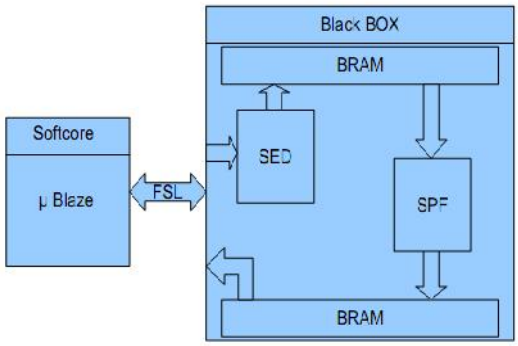

Figure 7: Hline architecture

Figure 7 represents the accelerator's architecture. The Microblaze, communicates with a black Box containing the two VHDL functions which write data on BRAMs.

The microblaze communicates with Black box through a FSL bus. The inputs of the black box are processed by the SED module. The SED results are saved in the first BRAM block.

This data are the input of the SPF module which fills all the triangles color in the second BRAM block.

The resulted data are sent via the FSL to the Microblaze to end the process. This data are sent for the VGA controller to display it.

We test this architecture with the 3D application. We do the test with an only one hardware accelerator then with 4 accelerators in parallel.

Aiming the AR applications, we'll not project the 3D object on the screen but a smallest representation of it on the screen.

3.2. Partially reconfigurable architecture and obtained results :

Now, we describe the proposed reconfigurable architecture and the obtained results. Instead of using a static architecture with a great number of frozen accelerators, we chose to make our approach better using a partially reconfigurable architecture.

This architecture will contain a number of accelerators which are not variable and another number of accelerators which depends on the type of the application.

3.2.1 Proposed architecture:

The adopted reconfigurable architecture is described in the figure 8. There are 3 hardware accelerator zones.

The zone 1 (Z1) contains Normal #1 and Transform #1 as described previously. These functions represented the geometric shader. This zone is a permanent one. This zone is attached to the microblaze [14] via FSL [19]. The data will be sent to the zone 2 or zone 3 for vertex shading. The microblaze is the Xilinx softcore.

The zone 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 barycentre 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 [14]. We can use also a picoblaze to do this task.

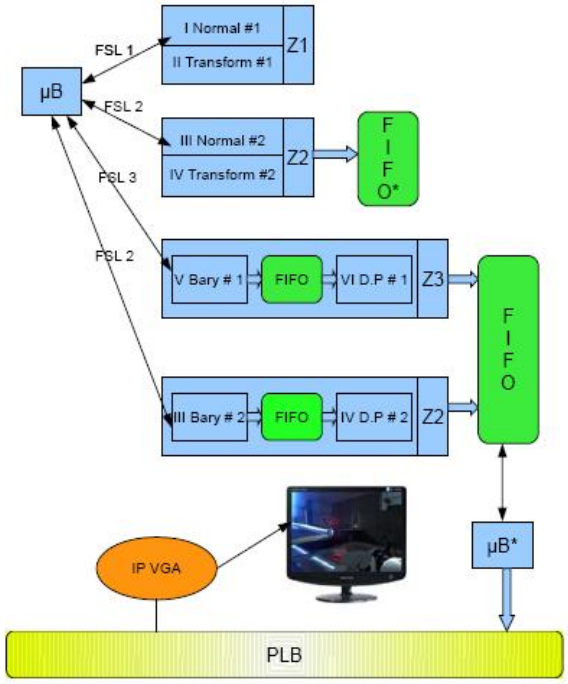

Figure 8: Reconfigurable proposed architecture

4 Obtained results:

In this part we'll talk about the obtained results of the adaptive HW/SW adequacy. We begin by using an adaptive approach based on adding many hardware accelerators for obtaining a better 3D displaying results.

4.1. Hline's accelerator :

4.1.1. Different divisions description:

In the hline function we need to use a HW division. We compare the both hardware accelerator with a pure software division. We've used a Xilinx IP. We have developed a division IP based on the processor division.

The latter proposed division agorithm is represented by the figure 9.

The proposed IP contains 5 states:

- ✓ IDLE: permits to initialise different signals.
- ✓ Sigcalcul : permits to know if it's a negative or positive division.
- ✓ Precalcul: permits to prepare to variables for calculation
- ✓ Calcul: permits to calculate the results.
- ✓ Endcalcul : permits to find the final division value.

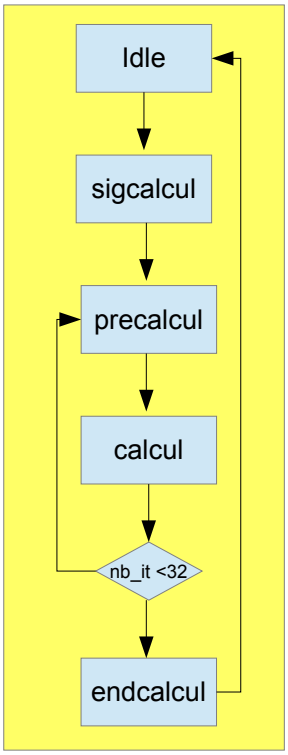

Figure 9: Proposed Division IP

The obtained results by the 3 different divisions are described by the Table 2.

Table 2: Division comparison

Division	Time	Nb of slices
Created	67 clocks	134
Xilinx	8 clocks	1320
Soft	668 clocks	

As shown in table 2, the created division is the best one in terms of time consumption and number of slices. In fact, using the Xilinx division makes the FPGA saturated. We will describe in the next part different division use in the IP generation.

4.1.2. Different divisions description:

The division IP developed by Xilinx is optimal in term of time execution. The major problem of this IP is that it consumes almost 1000 slices. That's why we can optimize the IP using Xilinx division IP or Created division IP or finally a mixed one. We describe the obtained results for the SED and the SPF parts.

4.1.2.1 SED division optimization:

According to the SED IP, we used 6 divisions. The use of the Xilinx one requires 8 clocks cycles but 1320 slices. The use of two accelerators with the

used architecture requires almost 25% of the FPGA surface. That's why we prefer to use an only one Xilinx division to not overcrowd it. When we operate with the created division IP, we can use 6 divisions with parallelism. This block will use only 1072 slices in 67 cycles. The data comparison is described in the table 3.

Table 3: SED comparison

Division	Clocks time	Nb of slices
6 created divisions	67 clocks	804
1 Xilinx division	64 clocks	1320

The use of the created divisions makes us lose 3 clocks cycles but we have a won of 516 slices. By consequence, we prefer to use the developed division IPs.

4.1.2.2 SPF division optimization:

According to the SPF IP, the required task needs to have a division for each line calculation. For each polygon, the number of lines belongs to the interval [15 120].

We used two architectures. The first one contains one Xilinx accelerator and the second one use 10 created accelerators in the same time. The obtained results are represented in the figure 10.

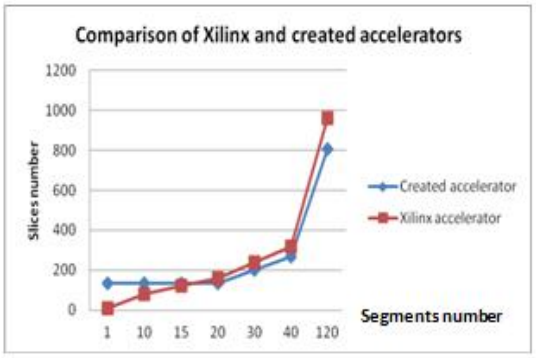

Figure 10: Division VHDL IP

If the used polygon contains less than 18 segments, we use the Xilinx division IP. If the polygon segments number is more than 18, the use of created division IP permits to have an optimal slice number.

4.1.3. Mixed SW/HW comparison :

The test of pure software 3D object code and the mixed Software/Hardware one is exposed in the Table 4.

Table 4: Software / Hardware comparison

Version	Time (s)	Frame/second (fps)
Software	6.5	0.15
Mixed HW/ SW		
1 HW Block	0.273	4
4 HW Blocks	0.075	14

The hardware accelerator allows a speed up of almost 24x. The use of multiple accelerators gives implementation almost 87 times faster than the software version. This improvement was possible due to the architectural parallelism. A frame rate of 14 fps is achieved with 4 HW blocks, with an acceleration of geometric functions and improvement of the current standard C software implementation the objective of 24fps will be obtained.

The FPGA occupation after the hardware implementation is described in the table 5. The use of 4 hardware blocks is 4 times harder to design. We increase the FPGA use to obtain a better result.

Table 5: Single Hline accelerator

Device utilization summery	Number	% ML 507
Slice Registers	5,790	12
Used memory	204	1
External Memory (kb)	1,080	20

We have presented our concept of flexible architecture for AR systems and detailed the main accelerator dedicated to pixel shading. The second accelerator, dedicated to geometric operations will be presented, in a following paper. The combination of both accelerators will guaranty a 24 fps frame rate with a 100MHz clock frequency. But, as previously mentioned, resource requirements are strongly data dependent in 3D applications. So, the

second step is the implementation of self-adaptation as a software function on the Microblaze that controls online the dynamic configuration of hardware accelerators, according to application needs. We've developed different accelerators using a VHDL code. We tested the different blocs independently.

Using an only one arithmetic accelerator we obtained a gain of 25% in time execution. Let's treat the rotation part and specially the square root VHDL development.

4.2. Square root VHDL coding :

In the rotation bloc, the square root bloc needs to be transformed. It's the harder and more consumption bloc. As we do for the division bloc, we create an sqrt VHDL bloc.

The blocs contained by the accelerator are:

- ✓ IDLE : Intial state.
- ✓ Nb_it : counter of the number of itereations.
- ✓ Prep_coeff: permits to prepare de used coefficients fot the bits detection of the number.
- ✓ Diff_1: permits to calculate the result if the the iteration number < 2.
- ✓ Prep_val : permits to find the data to use for Diff_2
- ✓ Diff_2: permits to prepare the result. The sqrt result is obtained when we have the max of nb_it.

The sqrt developped bloc is described by the figure 11.

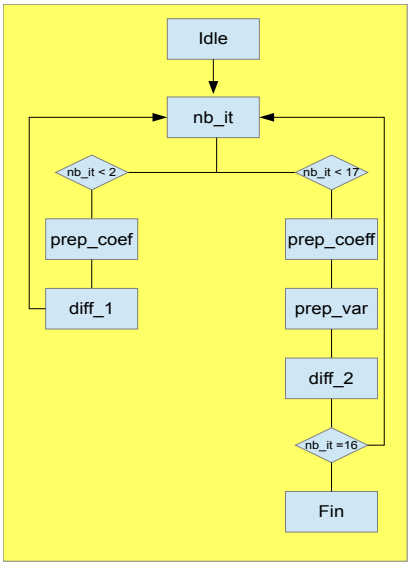

Figure 11: SQRT VHDL IP

After adding different IPs to the 3D application, the profiling obtained results are summarized in the table 6

Table 6: Single Hline accelerator

SQRT	Time	Slices
Created SQRT	64 clocks	40 slices
Xilinx SQRT	6 clocks	461 slices
Square root Soft	120 clocks	

As for the division, the use of the created SQRT provides a faster execution time by 50% while consuming only 40 slices. Comparing to the Xilinx solution, our solution is 10 times slower but has the advantage of using ten times less slices.

4.3. Partial dynamically reconfiguration approach :

In this part, we represent a partial dynamically reconfiguration approach obtained results.

Using the reconfigurable proposed architecture with different accelerators reconfigurable blocs we've obtained the results of the table 7.

Table 7: Accelerator bloc execution time

Used function	Execution time cycles	% gain
Normalize	2397997	70
Transformation	325676	40
Barycentre and dessine_poly	194244	70
Total	2917917	60

The table 7 that describes the accelerator blocks 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.

5 Conclusion and perspectives:

We have presented our concept of mixed HW/SW architecture using accelerators dedicated to the 3D application. The obtained results are very interesting and make clearer the steps to do for obtaining a better QoS. We adopt an architecture based on data adaptation and implement it on the Xilinx ML 507embedded kit. The obtained results are very positive and make easier the possibility of using this architecture in our augmented reality application.

According to the complexity of the application in term of texture or movement and our need to have a general approach that can accept any type of 3D objects.

The dynamically partial reconfiguration permits to optimize the used surface and the time consumption.

Our next goal consists of testing other applications with the used architecture such Open GL ES 3D application. We will use also the consumption measurement results for including the consumption constraint as an adaptation constraint.

References

[1] Kais Loukil, PhD, “Approche de gestion de performances/contraintes pour lessystèmes embarqués temps réel”, 2011

[2] Cemil Azizoglu, Ph. D, “High Performance Graphics on Android”, Khronos group, 2010

[3] 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.

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

[5] P. Fuchs, B. Arnaldi, and J. Tisseau. La réalité virtuelle et ses applications,chapter 1, pages 3–52. Les Presses de l’Ecole des Mines de Paris, 2003.

[6] P. Ngoc, G. Lafruit, J-Y. Mignolet , G. Deconinck, and R. Lauwereins “QOS aware HW/SW partitioning on run-time reconfigurablemultimedia platforms” Proceedings of the International Conference onEngineering of Reconfigurable Systems and Algorithms, ERSA'04, June21-24, 2004, Las Vegas, Nevada, USA. CSREA Press 2004, ISBN 1932415-42-4

[7] W. Van Raemdonck, G. Lafruit, E.F.M. Steffens, C.M. Otero Pérez, R.J.Bril “Scalable graphics processing in consumer terminals” Multimediaand Expo, 2002. ICME '02. Proceedings. 2002 IEEE InternationalConference

[8] Youenn Corre, Jean-Philippe Diguet, Dominique Heller, Loïc Lagadec: A framework for high-level synthesis of heterogeneous MP-SoC. ACM Great Lakes Symposium on VLSI 2012: 283-286

[9] A.Milakovich, V.Shankar Gopinath, R.Lysecky, J.Sprinkle, Automated Software Generation and Hardware Coprocessor Synthesis for Data-Adaptable Reconfigurable Systems.

[10] Joint Photographic Experts Group. JPEG 2000. Image Compression Standard, www.jpeg.org/ jpeg2000/

[11] Alarasighe, M Gordon, M. Karczmarek, J.Lin, D.Maze, R.M?Rabbah, W.Thies. Language and Compiler Design for streaming Applications. Intenational Journal of Parallel Programming, 32(2), 2005

[12] Impulse Accelerated Technologies. Impulse CoDeveloper, www.impulseaccelerated.com, 2010

[13] J. Flinn and M. Satyanarayanan, “PowerScope: A tool for profiling theenergy usage of mobile applications,” in Proc. of 2nd IEEE Workshopon Mobile Computing Systems and Applications, Feb. 1999.

[14] www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf, MicroBlaze Processor Reference Guide.