OpenMAX Compliant Heterogeneous Multimedia Embedded Plaftorm

D. Fuente, J. Barba, J. Dondo, F. Rincón, J. C. Lopez
School of Computer Engineering, University of Castilla-La Mancha
Ciudad Real, Spain
{David.Fuente, Jesus.Barba, JulioDaniel.Dondo, Fernando.Rincon, JuanCarlos.Lopez}@uclm.es

***Abstract*—In multimedia embedded systems, the performance of the interconnection system is key to meet the system requirements due to the hard timing constraints and large amount of data they handle.**

Since buses are a widely used communication infrastructure in SoCs, this paper describes a Hw/Sw multimedia platform that offers efficient, optimized and dynamically configurable communication mechanisms for bus based systems. Such optimization mechanisms are inspired in an open standard for multimedia systems, which has been adapted and ported to the special needs of embedded systems.

At the end, we provide a transparent, homogeneous and decentralized way for efficient data transfers between hardware and software components on this platform, with a minimum overload.

I. Introduction

In general, multimedia systems have to face with the processing of a large amount of data with hard timing constraints. In order to fulfill the demanded requirements, pieces of the system functionality have been traditionally implemented on electronic circuits. The result is the improvement of both, the throughput and performance of the system.

The first approaches dealing with hardware-based acceleration made use of ASICs (Application Specific Integrated Circuit) or DSPs (Digital Signal Processors). Lately, the emergence of high capacity reconfigurable devices such as FPGAs (Field Programmable Gate Array) is encouraging developers to use this technology for multimedia embedded systems. Some of the reasons that back up this trend are: short development/prototyping time, the ability to customize hardware, flexibility and reconfiguration facilities, the low cost compared with other programmable devices, etc.

However, there is a lack of standardization in many facets of multimedia embedded systems (e.g. data interchange mechanisms between the Hw and Sw elements in a system). This fact leads to ad-hoc implementations whereas the adoption of standards provides portable, flexible and reusable designs for embedded systems.

In this work we present a Hw/Sw platform for embedded multimedia systems, based on the use of the OpenMAX standard [1]. The goal is to provide the developers with the infrastructure and mechanisms that allow to build fully reusable and portable multimedia embedded systems in a reasonable time.

OpenMAX is an open standard, promoted by the Khronos Group, which pursuits to reduce the cost and complexity of porting multimedia software to new processors and architectures. Since the OpenMax reference implementation is only software, it has been necessary to revisit and redesign all the architectural concepts and protocols offered by OpenMAX in order to be applied to multimedia embedded systems. OpenMAX standard provides an API that enables porting both components and complete applications to multiple operating systems and platforms.

As mentioned previously, the proposed platform allows the development of Hw/Sw multimedia systems, and hence, a flexible and transparent mechanism to manage Hw/Sw communication is necessary. In addition, to cope with the temporal and computational requirements, it is mandatory to reduce the overload introduced by the integration infrastructure to a minimum. To get it, we rely on the facilities provided by the Object Oriented Communication Engine (OOCE) [2]. OOCE is an hybrid middleware, based on bus architecture, for SoCs. It provides basic and advanced in-chip communication services to transparently handle communication between the software and the hardware parts of an embedded system. We have extended OOCE with new features and some optimizations in order to meet the performance levels demanded by multimedia systems.

The rest of the paper is structured as follows: Section 2 gives an overview of the related works. In section 3 we offer a general view of OpenMAX and its application to the field of embedded systems. Section 4 analyzes our proposal and, in section 5 we explain a specialization of the communication mechanisms, based on a decentralized hardware-to-hardware data transfers and synchronization mechanism. Then, in sections 6 we describe the results of their application. Finally, section 7 summarizes the paper.

II. Related works

Multimedia applications have their own requirements that differentiate such applications from other domains. For example: an intensive traffic of data and use of the memory subsystem, a characteristic application model, real time restrictions, etc. These particular features have led the research community to invest a significant amount of effort in projects addressing the specific challenges imposed by the development of multimedia (embedded) systems.

The most widespread solution to improve the throughput relates to the development of the multimedia platform with hardware acceleration in mind. This solution is mentioned in [3] and [4], which respectively implement the computational cores as coprocessors or heterogeneous reconfigurable engines.

The Cell BroadBand Processor [5], with its characteristic four ring buses, combines a general-purpose PowerPC architecture with streamlined coprocessing elements which greatly accelerate multimedia and vector processing applications.

Recently, multi-core architectures for graphical applications, such as NVidia GPUs arrays, have gained a great importance mainly due to the availability of CUDA [6], an open software development framework based on a standard widespread language as C.

PeaCE [7] is another alternative to develop Hw/Sw multimedia platforms. PeaCE provides a development flow (from functional simulation to the synthesis process) to multimedia applications with real time constraints.

An important feature of these platforms is how the components in the system communicate. Generally, communication mechanisms are fixed, with a limited capability to be customized. Since there are many factors that can affect to the communication performance that should be considered, offering the ability to reorganize the communication channels in run time is essential. This would also be a useful tool for designers that could quickly and easily perform a design space exploration.

For example, Na Ra Yang et al. [8] propose double buffer, open row access and interleaved memory techniques in order to achieve an efficient communication. Another alternative to improve the communication performance is described in [9] which presents a solution based on a time multiplexed memory, where the system memory is accessed several times during a single bus cycle. To allow this, the memory and the DMA logic must operate at a clock frequency which is a multiple of the clock frequency of the microprocessor.

In relation to the communications performance analysis in multimedia dedicated platform, in [10] several alternatives for a shared memory data exchange mechanism are compared: point-to-point connection, and based on bus-based architecture using DMA. In the final part of the work, they present a performance estimation tool based on a Bus and Synchronization Event Graph (BSE graph) that is used to obtain statistical results.

III. OpenMAX Multimedia Application

A typical multimedia application in OpenMAX is formed by a chain of processing elements called OpenMAX Components (OMX Component from now) that process data on their inputs, generating new data on their outputs. An OMX Component implements one or more media processing functions out of four application domains (audio, video, image and other) as defined in the standard. The OMX Integration Layer (IL), [11]) abstracts and unifies the access to the functionality embodied in the OMX Components and also provides different functions to: establish communications between OMX Components, send commands to the OMX Component, configure the OMX Component parameters and obtain the necessary resources.

The OMX Components in an application exchanges media data in packets called *buffers*. The communication is handled through ports which can be classified as an input or output port, depending on whether it receives or sends buffers. The standard describes three types of communication:

- Tunneled: the communication between components takes place directly without the intervention of any other component in the system. To this end, the standard defines a buffer exchange convention.
- Non-Tunneled: the communication between components takes place through the entity that implements the application control.
- Proprietary: the communication between components takes place directly but, opposite to the tunneled case, the mechanism is not defined by the standard.

The IL houses some of the most important functionality in OpenMAX (initialization and connection of the OMX Components and resource communication and synchronization management). Therefore a hardware implementation of the main OMX Components will help to raise the throughput of an OpenMAX-based multimedia platform.

A. OpenMAX Hardware Implementation

The implementation process to adapt OpenMAX to an embedded multimedia heterogeneous environment, started with the identification of the critical parts of the standard that should be implemented in silicon. Later, we performed an efficient Hw implementation of the entities and communication methods selected using the facilities provided by OOCE.

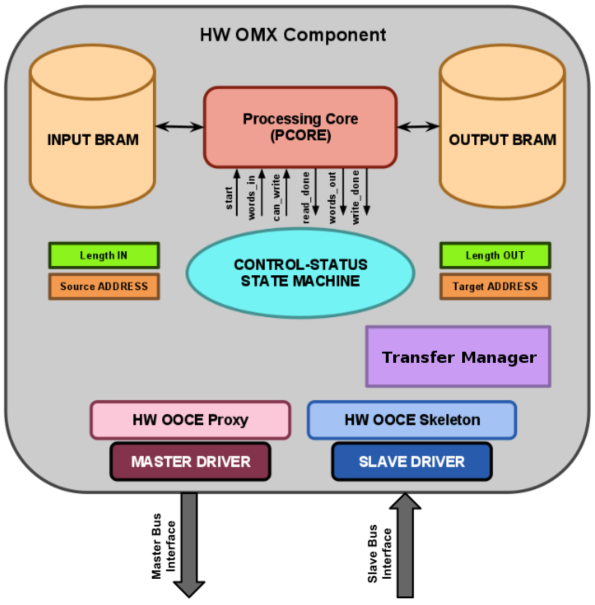

Fig. 1. Hw OMX Component structure

The most important element of the subset implemented in Hw is the OMX Component (Hw OMX Component) which belongs to the IL layer. This element encapsulates the hardware module, the *Processing Core* (PCore), that actually implements the multimedia function. The PCore interacts with the rest of the logic through a fixed physical interface, facilitating, thereby, its reuse. The architecture of the Hw OMX Component, which is generated automatically, is thought as the placeholder for the PCore and it is mainly dominated by the presence of two local memories that are used to store the input and output buffers to be produced or consumed by the PCore.

The Transfer Manager (TM) is the subsystem, within the OMX Component, responsible for the buffer exchanging process between one Hw OMX Components and any other regardless of its implementation (Hw or Sw). The TM realizes the concept of *communication ports* in the standard. In order to establish the communication with the previous/following OMX Component in the chain, the TM uses the references (OOCE proxies) contained in the *source* and *target* registers. The TM implements the three communications types described in the OpenMAX because the target application could benefit from the variety of the communication models provided and the flexibility introduced in the design.

Although two Hw OMX Components can communicate in three different ways (tunneled, non-tunneled and proprietary), the most optimal mechanism to exchange buffers is the tunneled communication because the synchronization, control and data flow traffic is handled directly by the two actors in the communication process. For this reason, in this paper we focus on the adaptation made of tunneled communication. In our implementation, this means this process takes place between two hardware instances of an Hw OMX Component without the intervention of a controller or any kind of software process. This has a positive impact in performance since the processor is not mediating in every single data transfer that takes place in the system. The OOCE Hardware-to-Hardware invocation semantics makes it possible. Moreover, the use of OOCE in our proposal enriches the final solution, adding transparency and homogeneity to the platform.

Finally, the Control Unit (CU) dispatches the OpenMAX commands coming from communication layer to the correct entity in the component for further processing. Also, the CU orchestrates the internal communication of all the subsystems in the OMX Component. The OpenMax commands are sent and received through the OOCE adapters (*proxies, skeletons and drivers*) that make the Hw OMX Component independent of the bus technology and, therefore, the platform in which it is going to be deployed. OOCE establishes the semantics to guarantee transparent interoperability between any two OMX Components (either Hw or Sw).

IV. The proposal

In this proposal, we rely on the use of FPGAs devices and industry standards for the design and development of multimedia systems which allow portable and efficient implementations of in less time.

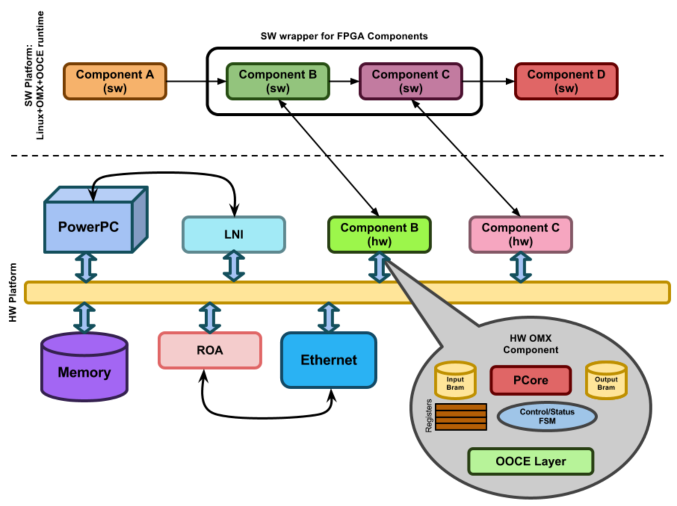

Fig. 2. Implementation example of an heterogeneous hw/sw multimedia processing chain with HW OpenMax Components.

As a global view of the proposal, in figure 2 it is represented how a multimedia processing chain is implemented using our proposal. In this example, components B and C in the chain are mapped to a hardware implementation of an OMX Component and they can communicate using any communication models offered by OpenMAX. Communication between two OMX Components that remain implemented in Sw is not treated here since it follows the same principles and mechanism as defined in the documentation of the standard and the implementation reference provided by Khronos. Therefore, in this section we focus on those scenarios where a Hw version of the OMX Component is present.

Every Hw OMX Component has a software counterpart that interacts with the rest of the OpenMax middleware. This is mandatory in order to keep the rest of the application unchanged. This software wrapper stands for the hardware component and exactly implements the same API required by the standard, acting as a facade.

To illustrate what *transparency* and *easy integration* of Hw OMX Components actually mean, let us introduce the listing 1 showing a fragment of an OpenMax application running in a FPGA. The application receives one video frame from the Ethernet interface, transforms the RGB picture to grey scale, applies a border detection algorithm and finally sends the result via Ethernet. The *img_ethreader* and *image_ethsink* components are in charge of the acquisition of the image to be processed by the chain, and the delivery of the result respectively. These components run in software, whereas the *hw.img_RGB2BW* and *hw.img_sobel* components are implemented in hardware. Nonetheless, there are no distinctions between the use of the hardware components and the software ones since both of them are OMX compatible.

```
int main(int argc, char** argv) {
/*Getting Components Handler*/
OMX_GetHandle(&appPriv->ethreader, ''omx.ext.image.
      ethreader'', NULL,&readercallbacks);
OMX_GetHandle(&appPriv->hwrgb2bw, ''omx.hw.
      img_RBG2BW'', &RGB2BWcallbacks);
OMX_GetHandle(&appPriv->hwsobel, ''omx.hw.img_sobel'
      ', NULL, &hwsobelcallbacks);
OMX_GetHandle(&appPriv->ethsink, ''omx.ext.image.
      ethsink'', NULL, &sinkcallbacks);
/*Set the size for img Hw OMX Components*/;
sSize.sWidth.nValue = 640;
sSize.sHeigth.nValue = 480;
OMX_SetParameters(&appPriv->hwrgb2bw,
      EXT_OMX_IndexConfigImgSize, &sSize);
OMX_SetParameters(&appPriv->hwsobel,
      EXT_OMX_IndexConfigImgSize, &sSize);
/*Setting up tunneled communication*/
OMX_SetupTunnel(appPriv->hwrgb2bw, 1, appPriv->
      hwsobel,0);
/* Change HW OMX Component state */
OMX_SendCommand(appPriv->hwrgb2bw,
      OMX_CommandStateSet, OMX_StateIdle, NULL);
OMX_SendCommand(appPriv->hwsobel,
      OMX_CommandStateSet, OMX_StateIdle, NULL);
OMX_SendCommand(appPriv->hwrgb2bw,
      OMX_CommandStateSet, OMX_StateExecuting, NULL);
OMX_SendCommand(appPriv->hwsobel,
      OMX_CommandStateSet, OMX_StateExecuting, NULL);
...
OMX_DeInit();
return 0;
 }
```

Listing 1. Example of an implementation of an heterogeneous Hw/Sw multimedia processing chain with Hw OpenMax Cores.

Behind the scenes, there is much more to take into account for hardware components. For example, figure 3 represents the sequence of OOCE messages needed to perform some configuration on the Hw OMX Component. When the OMX Component, that is used as a facade, receives certain standardized notifications, it redirects the notifications to the associated Hw OMX Component via messages. These messages are translated into bus transactions by OOCE middleware and are collected by de Hw OMX Component through the OOCE adapters. For example, when a OMX Component receives the configuration values by OMX_SetParameters() notification, it sends two messages to the associated Hw OMX Component in order to configure the buffers length and the specific configuration registers.

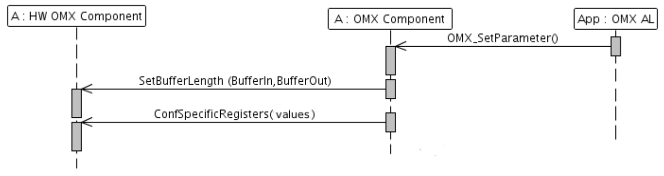

Fig. 3. Sequence diagram of the Hw OMX Component configuration messages.

Similarly, when the OMX Component receives the notifications to establish a tunneled communication or to start execution, it transforms and extends these notifications to the Hw OMX Component. Once configured and started, two Hw OMX Components can begin to exchange buffers following the OOCE Hardware-to-Hardware invocation semantics.

V. Hardware-to-Hardware buffers exchanging strategy

As previously mentioned, tunneled communication is the most efficient buffer exchange scheme offered by the standard since it reduces the synchronization traffic. The producer and the consumer use a rendezvous protocol with no intermediaries nor controllers. Therefore, it is quite interesting its application in an embedded system for the best use of the available communication bandwidth.

However, a Hardware-to-Hardware dialog is rarely present in heterogeneous embedded platforms which prevents an optimal implementation of the standard. But this is not the case of the OOCE SoC middleware which allows two IPs to establish direct communication via hardware remote method invocations.

On top of this, we have implemented a buffer exchange convention and synchronization mechanism between Hw OMX Components inspired in the standard. The communication meets the producer/consumer pattern and the components, involved in the communication, exchange the usual data traffic besides two types of synchronization messages ("FillBuffer" and "EmptyBuffer"). Again these messages are translated into bus transaction.

The main idea in Hardware-to-Hardware communication is to transfer as soon as possible the content of a buffer content between two Hw OMX Components without the intervention of any other component in the system. The sequence of steps (illustrated in figure 4) in a Hardware-to-Hardware communication is the following:

1) At the beginning, the input local memory of the consumer Hw OMX Component is empty so sends a "FillBuffer" message to the component playing the role of producer and wait.
2) When the producer Hw OMX Component output memory is full, the producer writes the buffer content into consumer's input memory and sends a "EmptyBuffer" notification.
3) When the consumer has read the last word from the buffer in its input memory (not necessarily processed) it sends a "FillBuffer" notification to the producer again, avoiding unnecessary waiting.

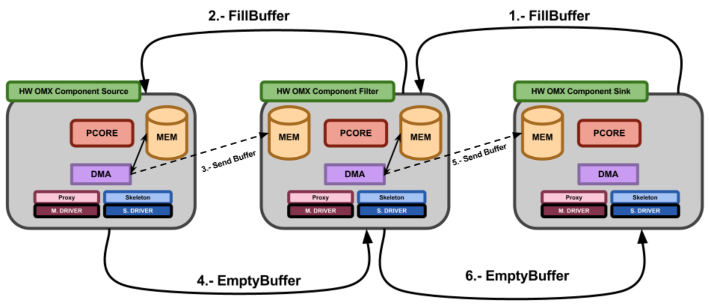

Fig. 4. Sequence of messages in Hw-to-Hw Communication.

Over this general communication model, we have developed two different optimizations in order to reduce the transmission latency and maximize the use of the available bandwidth. With the proposed optimizations, the number of wait cycles, due to peaks in the traffic load, is reduced, re-distributing the bus workload using techniques that parallelize and interleave all the transfers between OMX components. This is achieved by means of using the dead times while the component is processing a buffer.

The first optimization (see Figure 5) forces the Hw OMX Component to transmit a buffer in the output memory before filling it. As soon as there is a packet of N-words (configurable via an internal register), it is transmitted. This optimization is called "No Wait Until Fill" (NWUF). The benefit of using this technique is twofold: (1) component's execution and output buffer transfer are overlapped; and (2) several buffer transmissions can take place at the same time.

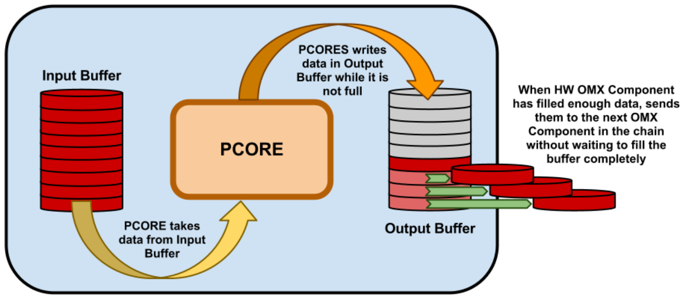
Input Buffer
PCORES writes data in Output Buffer while it is not full
When HW OMX Component has filled enough data, sends them to the next OMX Component in the chain without waiting to fill the buffer completely
PCORE
PCORE takes data from Input Buffer
Output Buffer

Fig. 5. No Wait Until Full optimization

The second optimization (see Figure 6) is called MBF ("Multiple buffering") and allows to overlap: (1) the reading of the actual input buffer with the writing of a new input buffer; and (2) the transmission of the actual output buffer with the writing of a new output buffer. To this end, the physical address space of one memory is logically divided into N independent regions which are managed separately. Thus, full parallelism is achieved.

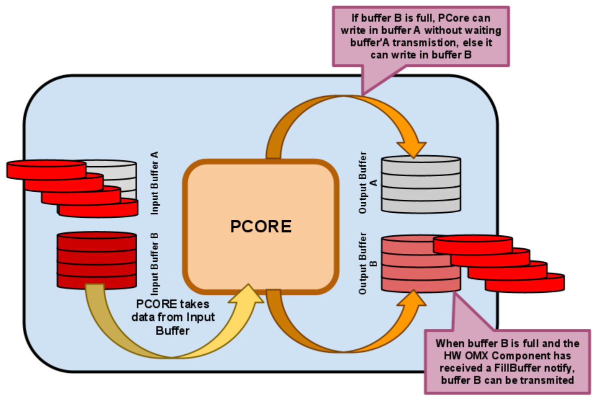
If buffer B is full, PCore can write in buffer A without waiting buffer'A transmistion, else it can write in buffer B
Input Buffer A
Output Buffer A
PCORE
Input Buffer B
Output Buffer B
PCORE takes data from Input Buffer
When buffer B is full and the HW OMX Component has received a FillBuffer notify, buffer B can be transmited

Fig. 6. MultiBuffer optimization example

Both optimizations are compatible and they can be used simultaneously but, depending on the case, their effectiveness might vary. This is going to be analyzed in the next section.

In any case, several communication parameters are customizable for our platform: the size of the buffer, the packet length, the number of components connected to the bus and the size of the data transmitted.

VI. Experimental results

In order to provide the reader with a glimpse of the benefits and efficiency of the proposed approach, two experiments are dissected in this section. Particularly, we have focused in the analysis of the NWUF and MBF techniques and how they help to increase the system performance compared against the base case (Wait Until Fill and No Multiple Buffers).

The prototyping platform used in the experiments was a Xilinx Virtex5 based board with a MicroBlaze running at 100Mhz and a 64 bit PLB bus. All the Hw OMX Components have two local memories whose sizes are 1 Kword. Several tests were generated and downloaded to the board with a variable number of hardware and software components. For each test, 1Mword of synthetic data was fed into the processing chain.

It is worth mentioning that in this case we used PLB, but it is almost straightforward to perform new test over other platforms just changing the bus type.

The first picture (figure 7) illustrates the benefits obtained by applying the "No Wait Until Full" optimization mechanism (described in section V). As it is expected, the occupancy of the bus increases with the number of components in the processing chain. What is remarkable is how our proposal scales in a linear pattern. The measured transmission times are reduced in 20% (mean value) for the six cases when compared with the "Wait Until Full" strategy.

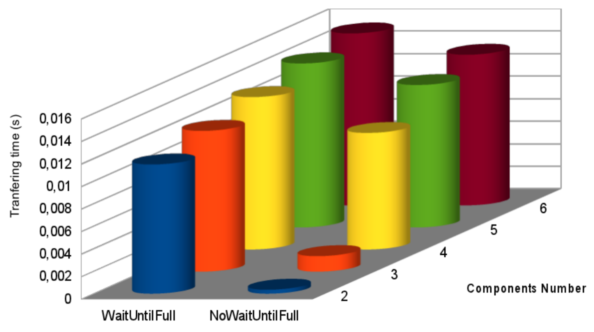

Fig. 7. Comparative chart of transferring time based on the number of components and "No Wait Until Full" optimization.

The second picture (figure 8) represents how the size of the buffer has an influence on the total transmission time. The benefits of the "Multiple buffering" technique are exposed in this experiments and its applicability is delimited, identifying such scenarios where its application makes sense.

Since the size of the local memories is 1Kword, the number of buffers that can be held in it varies (for example, two buffers for a 512 words buffer-size configuration). In this chart, we can see what is the best value for a certain buffer size configuration. To increase the number of buffers means that the control logic also increases, so it is necessary to estimate the cost/performance relation.

From the picture, two lesson can be learnt: (a) in systems with a low overhead regarding the bus bandwidth requirements, the bigger the size of the buffer the better the performance of the system; whereas (b) in the opposite situation it is better to reduce the size of the buffer since several buffers transmissions can be interleaved, reducing the wait times.

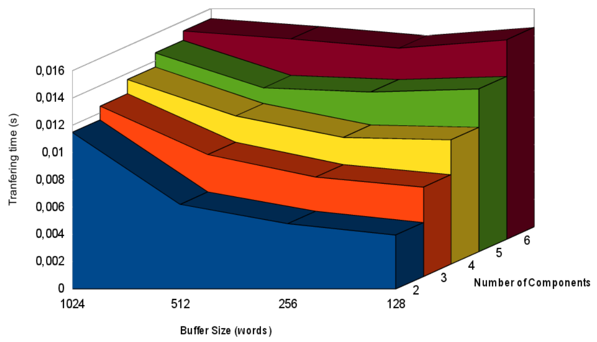

Fig. 8. Comparative chart of transferring time based on the number of components and "Multiple buffering" technique.

VII. Conclusion

Multimedia applications need of specialized embedded systems in order to deal with the demanding future it is discerned for the next years. This work offers an open, flexible solution based on accessible technologies and devices but without sacrificing performance and efficiency. To this end, we have presented a platform with a communication strategy for heterogeneous multimedia embedded system with a set of parameters and optimizations, based on OpenMAX standard and OOCE.

References

[1] (2011, September). The Khronos Group website. Available:http://www.khronos.org/openmax/

[2] J. Barba, F. Rincón, F. Moya, J.D. Dondo, F.J. Villanueva, D. Villa, J.C. López, *OOCE: Object-Oriented Communication Engine for SoC Design* DSD - Euromicro Conference on Digital System Design. Lübeck (Germany), 2007.

[3] Martijn J. Rutten, Jos T.J. van Eijndhoven, Evert-Jan D.Pol, Egbert G.T. Jaspers, Pieter Van der Wolf, Om Prakash Gangwal and Adwin Timmer. *"Eclipse: A Heterogeneous Multiprocessor Architercture For Flexible Processing"*. Philips Research Laboratories, 2002.

[4] Paul Brelet, Arnaud Grasset, Philippe Bonnot, Frank Ieromnimon and Dimitrios Kritharidis. *"System Level Design for Embedded Reconfigurable Systems using MORPHEUS platform"*. IEEE Annual Symposium on VLSI, 2010.

[5] Kahle J. A., Day M.N., Hofstee H. P., Johns C. R., Maeurer T. R. and Shippy D. *"Introduction to the Cell Multiprocessor"*. IBM Journal of Research and Development. 2005.

[6] Buck, Ian. *"GPU computing with NVIDIA CUDA"*. ACM SIGGRAPH'07. 2007. San Diego (California).

[7] Soonhoi Ha, Sungchan Kim, Youngming Yi, Seongnam Kwon and Youngpyo Joo.*"PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems"*. ACM Transactions on Design Automation of Electronic Systems, August 2007.

[8] Na Ra Yang, Gilsang Yoon, Jeonghwan Lee, Intae Hwang, Cheol Hong Kim, Sung Woo Chung, Jong Myon Kim, *Improving the System-on-a-Chip Performance for Mobile Systems by Using Efficient Bus Interface* cmc, vol. 2, pp.606-608, 2009 WRI International Conference on Communications and Mobile Computing, 2009

[9] C. Brunelli, F. Garzia, C. Giliberto, and J. Nurmi, *A dedicated DMA logic addressing a time multiplexed memory to reduce the effects of the system bus bottleneck*, in Proc. FPL, 2008, pp.487-490.

[10] Lahiri, Kanishka and Raghunathan, Anand and Dey Sujit, *Fast performance analysis of bus-based system-on-chip communication architectures*. Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design (ICCAD '99)

[11] OpenMAX Integration Layer Application Programming Interface Specification. Version 1.1.1. 2007. The Khronos Group Inc.