The CO-Simulation Interface SystemC/Matlab Applied IN JPEG Algorithm

Walid Hassairi
Laboratory CES, National School of
Engineers of Sfax, Tunisia.
walid.hassairi@ceslab.org

Moncef Bousselmi
Laboratory CES, National School of
Engineers of Sfax, Tunisia.
moncef.bousselmi@ceslab.org

Mohamed Abid
Laboratory CES, National School of
Engineers of Sfax, Tunisia.
mohamed.abid@ceslab.org

***Abstract*— Functional verification is a major part of today's system design task. Several approaches are available for verification on a high abstraction level, where designs are often modeled using MATLAB/Simulink. However, different approaches are a barrier to a unified verification flow. In this paper, we propose a co-simulation interface between SystemC and MATLAB and Simulink to enable functional verification of multi-abstraction levels designs. The resulting verification flow is tested on JPEG compression algorithm. The required synchronization of both simulation environments, as well as data type conversion is solved using the proposed co-simulation flow. We divided into two encoder jpeg parts. First implemented in SystemC which is the DCT is representing the party HW. The second consisting of quantization and entropy encoding is implemented in Matlab is the SW part. For communication and synchronization between these two parts we use S-Function and engine in Simulink matlab. With this research premise, this study introduces a new implementation of a Hardware SystemC of DCT. We compare the result of our simulation compared to SW / SW. We observe a reduction in simulation time you have 88.15%.**

Keywords: hardware/software, co-design, co-simulation, systemC, matlab, S-Function, communication, synchronization, jpeg, stimulus.

I. INTRODUCTION

The functionality of embedded systems as well as the time-to-market pressure has been continuously increasing in the past decades. Simulation of an entire system including both hardware and software from early design stages is one of the effective approaches to improve the design productivity. A large number of research efforts on hardware/software (HW/SW) co-simulation have been made so far. Real-time operating systems have become one of the important components in embedded systems. However, in order to validate the entire system functionality, this system has to be simulated together with application software and hardware. Indeed, traditional methods of verification have proven to be insufficient for complex digital systems. Register transfer level test-benches have become too complex to manage and slow to execute. New methods and verification techniques began to emerge over the past few years.

Our work articulates on three contributions which are the proposal for solutions to the implementation of the different parts of the architecture using SystemC and Matlab/Simulink simulators, the definition of a co-simulation environment based on the automatic generation of the interfaces required to the integration of these simulators and the proposal of a new verification framework based on SystemC Verification standard that uses MATLAB/Simulink to accelerate the testbench development. The MATLAB/Simulink to SystemC interface and the advanced version of transactors are combined in a scalable multi-abstraction level verification platform. The proposed refined co-simulation platform enables co-simulation with hardware models written in SystemC. On that platform, application software and hardware modules are directly executed on a host computer, which leads to a high co-simulation speed. The MATLAB/SystemC interface is mainly used for the verification of the lower abstraction levels with a high level model of their execution environment.

The integration of SystemC within MATLAB/Simulink and the resulting verification flow is tested on the JPEG compression algorithm. The required synchronization of both simulation environments, including data type conversion, is solved by using the proposed co-simulation flow. The application is divided into two JPEG encoder parts: the DCT (Direct Cosine Transform), the HW part implemented in SystemC, and the QEE (Quantization and Entropy Encoding), the SW part implemented in Matlab. With this research premise, this study introduces a new HW implementation of the DCT algorithm in SystemC.

In this paper, the related work is discussed in Section 2 and the proposed co-simulation

methodology is presented on Section 3. Then, in Section 4, we propose the implementation of the JPEG image compression as a case study. On Section 5, we summarize the proposed approach and co-simulation results. Finally, we conclude the proposal including suggestions and recommendations to future works.

II. RELATED WORK

Connecting Simulink and SystemC together has already been tried in the literature. Authors in [6] propose a solution to integrate SystemC models in Simulink. A wrapper is created using S-Functions to combine SystemC modules with Simulink.

This wrapper initializes the SystemC kernel and converts Simulink data type to SystemC signals and vice versa. Simulation control is entirely handled by Simulink. Some extensions of the SystemC kernel are required for initialization and simulation tasks. In [7], SystemC calls MATLAB using the engine library. MATLAB provides interfaces to external routines written in other programming languages. Using the C engine library, it is possible to share data between SystemC models and MATLAB. This simple working demo shows how to use the library to send and retrieve data from the MATLAB workspace and plot some results. The main difference with [6] is with the simulation control: SystemC is now the master of the simulation and MATLAB operates as a slave process. Also, Simulink is not supported in this example.

In a similar way, MathWorks provides a commercial solution to close the gap between algorithmic domain and the hardware design. The link for ModelSim [8] is a co-simulation interface that integrates MATLAB and Simulink into the hardware design flow. It provides a link between MATLAB/Simulink and Model Technology's HDL simulator, ModelSim. This interface makes the verification and co-simulation of RTL-level models possible from within MATLAB and Simulink. As opposed to the two previous techniques, there is no support for system level languages like SystemC.

These approaches [6, 7, 8] all try to reduce the barrier that exist between higher level modeling and existing hardware design flow. While [8] is a fully functional commercial tool for RTL verification, [6, 7] suffer from their embryonic stage (i.e. incomplete solutions for hardware design and verification).

The authors in [9] look at the problem of co-simulating continuous systems with discrete systems. The increasing complexity of continuous/discrete systems makes their simulation and validation a demanding task for the design of heterogeneous systems. They propose a co-simulation interface based on Simulink and SystemC. The main objective of the proposed solution is to provide a framework to evaluate continuous/discrete systems modeling and simulation.

In our former work [10], we adopted the methodology of communication and synchronization. To exchange data between a Simulink model and SystemC module, the co-simulation interface must integrate a bridge between the two simulators. This bridge is built with two Simulink S-Functions. An S-Function is a computer language description of a Simulink block. It uses syntax of call allowing us to interact with Simulink solvers. For our bridge, we create two C++ S-Functions.

The representation of simulation time differs significantly from SystemC and Matlab. SystemC is cycle-based simulator and simulation occurs at multiples of the SystemC resolution limit. The default time resolution is one pico-second. This limit can be changed with the function sc_set_time_resolution. However, Simulink maintains simulation time as a double precision value scaled to seconds. Thus, our co-simulation interface uses a one-to-one correspondence between simulation time in Simulink and SystemC.

The system architecture design in Simulink imposes some limitations and constraints. These design rules include some constraints. first constraint is on the selection and configuration of the blocks used for the application in traitement of signial., second contraint is on the integration of application functions implemented in other programming languages such as C/C++and final contraint is the construction of the system architecture model. The system architecture model may use only discrete Simulink blocks in order to allow a discrete event simulation. The constraint on the S-Functions is defined in the system architecture model. it may include user defined functions written only in C programming language. The S-Functions used to integrate the customized code need to be built by using the S-Function Builder tool, which is the fastest and easiest way for the SFunction generation compared with the manual implementation.

III. METHODOLOGIES

The implementation of applications on embedded systems is a very time expensive task using the standard development tools. The new heterogeneous model is executable too to simulate the co-design implementation. The simulation of the heterogeneous model is realized using SystemC. A description of a hardware module is transformed into a structural description with SystemC components (RT-level). The interface between hardware and software parts is implemented using special SystemC constructs and can be compared with the interface of the implementation in the real system. SystemC provides several levels of

abstraction to describe hardware. For the simulation of hardware modules in the shown design flow the cycle accurate level (CA) of SystemC is used. The interface to the software kernel is untimed functional level (UTF). A wrapper was designed to connect the modules to the software kernel. This wrapper is based on two shell-blocks which connect the CA-model to the software kernel by realizing an interface between the CA- and the UTF-model (Untimed Functional) of SystemC.

Simulink is a commonly used tool for designing DSP applications. It supports with a lot of libraries distinguished suppositions to develop single machine vision operators, e.g. the possibility to generate intelligent test environments for image.

To use the tool for generation of hardware operators, an interface between SystemC and Simulink was developed.

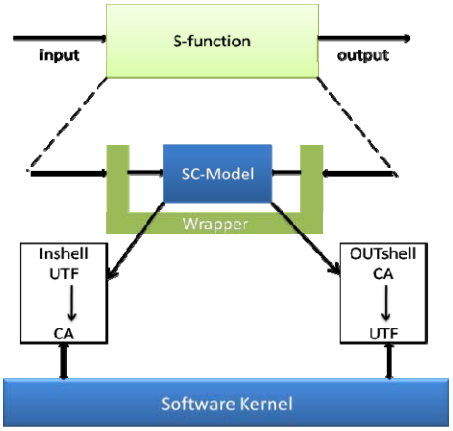

Figure 1: Using SystemC within Simulink S-Function

So, our methodology tries to push the idea a step further than just a co-simulation interface. It is a complete verification solution. It uses MATLAB external interfaces, similar to the example described in [6], to exchange data between SystemC and Simulink. Once this link is established, it opens up a wide range of additional capability to SystemC, like stimulus generation and data visualization [10]. The first advantage of our technique is to use the right tool for the right task. Complex stimulus generation and signal processing visualization are carried out with MATLAB and Simulink while hardware verification is performed with SystemC verification standard. The second advantage is to have a SystemC centric approach allowing greater flexibility and configurability.

There are three new call-backs provided via virtual methods for classes derived from sc_module, sc_port, sc_export, and sc_prim_channel. These call-backs will be invoked by the SystemC simulation kernel when certain phases of the simulation process occur.

In what follows, the modelling of the tasks in SystemC will be explained before describing the various manners admitted to transform the tasks of transactional Simulink into tasks described in SystemC.

A. Description the tasks in SystemC

For the modelling of the tasks in SystemC, we used the notion of "SC_MODULE". A module can be hierarchical containing the other modules, or elementary containing an active or passive behavior using the elementary modules "SC_CTHREAD".

```
SC_MODULE (SF_SYNCRO)
{
va_in_mac_pipe<long int> DATA_IN1;
va_in_mac_pipe<long int> DATA_OUT1;

va_synchro  TSAP2;
void SF_SYNCRO_beh();
SC_CTOR(SF_SYNCRO)
{
SC_CTHREAD( SF_SYNCRO_beh,TSAP2.pos())
};
};
```

Figure 2: Example of a file header. "h" has a corresponding TASK SystemC.

On the other hand, the communication is determined through an interface of communication. This last one is described through a set of ports which can be inputs, output or inputs / output ones. SystemC also supplies a specific port for the modelling of a physical clock.

```
#include <stdio.h>
#include <stdlib.h>
#include"SF_SYNCRO.H"
void SF_SYNCRO::SF_SYNCRO_BEH()
{
    long int entreel;
    long int sortiel;
        for(;;)
        {
            entreel= DATA_IN1.Get();
            //calcul
            DATA_OUT1.Put(sortiel);
        };
}
```

Figure 3: Example of a file header. "cpp" has a corresponding task SystemC.

The figure 2 shows the header file of a task described in SystemC. The interface of this module is formed by an input port and an output port of type 'long int'. The task has a service port 'SAP', which allows synchronization of tasks in the co-simulation. However, the figure 3 shows the main file"cpp". The main calculation is done to the body of this task .the communication of this with the system is through the interfaces represented by the ports of entry and exit 'DATA_IN1' and 'DATA_OUT1' by means of APTs defined in the library.

B. Transformation the S-Functions of Simulink in task SystemC.

SystemC is used by the synthesis tools and co-simulation in the stream of conception flow of the proposed heterogeneous Systems. The S-Functions are developed in language C according to precise rules and through methods decided by the Simulink simulator. An S-Function is formed by four essential

methods. In our work, a block S-Function will be converted in a module in SystemC trained by a ' thread ' sensitive to a signal ' SAP '. The file S-function C will be processed in a direct manner in a header file and implementation file in C + +.The Figure 4 presents an example of transformation of a block S-Function in a module SystemC. To understand better the transformation of one S-Function into a task, we decompose in four parts.

Part I: In this section, we define global variables and we include the header files. 'H'.
- S-function: header files of the library of Simulink (Simstruct.h ...) macros, header files of the code, and global variables are defined.

- SystemC: The header files of the SystemC library, macros, code header files and global variables are defined.

Part II: The initialization of variables and definition of input ports and output are included in this section.

- S-function: This part is formed by the method mdlInitializeSizes (SimStruct * S) where variables are initialized, and the number and size of ports of entry and exit are defined.
- SystemC: This part is divided on the header file and implementation file for SystemC. In the first type of port is defined. In the second module ports are declared and initialized. The type of the port depends on the type of communication used by the port (Shared memory, FIFO, signal synchronization).

Part III: The APIs and communication are the main calculation developed in this part along a loop that is repeated several times.

- S-function: Method mdloutput (SimStruct *S) is used in this part. The main calculation of the block is made. The data to be transmitted are affected ports by using the operator "=". This is a communication primitive.

- SystemC: The loop for (;;) in the implementation file contains the main calculation module. The calculation code in C is similar to that of the S-function.

The difference in this part occurs at the level of communication primitives. In S-function, a reading and writing data port is through the assignment operator "=". In SystemC there are two types of communication primitives:

- The Get () and Put () to communicate through a FIFO.

- The operator "=" to read and write to shared memory.

Part IV: This is the last part that runs at the end of the simulation.

- S-function: This part is formed by the method mdlterminate (SimStruct * S).

- SystemC: This part is after the end of the loop for (;;) of Part III and the end of the module.

The transformation of a task s-function in SystemC is semi-automatic. It is done for each S-Function to the application to develop.

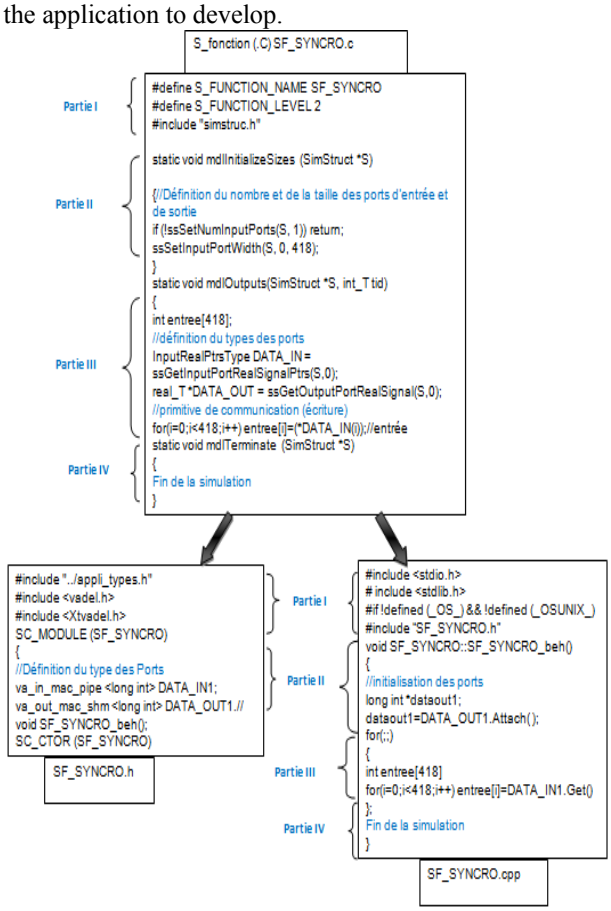

Figure 4: The transformation of the S-Function in SystemC.

IV. JPEG ALGORITHM

The process of JPEG-based encoding and decoding of images vary according to color depth (8, 24 or 32 bits). For an 8-bit image, this is simply one byte (8-bits) per pixel and for a 32-bit image; they are 4 bytes per pixel. The compression process for each block goes through the following processes in addition to preprocessing.

- Discrete Cosine Transform (DCT)
- Quantization
- Zigzag
- Entropy encoding (commonly Huffman)

Decompression is an inverse process that performs the individual inverse of all the above processes.

(i) DCT:

Discrete Cosine Transform is the heart of JPEG compression and also time consuming process. The following equations are the idealized mathematical definitions of 8x8 DCT:

$$F(u,v) = 1/4C(u)C(v)[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos((2x+1)u\pi)/16 * \cos((2y+1)v\pi/16]$$

Where:

$C(u)=C(v)=\frac{1}{\sqrt{2}}$ for u,v=0
C(u)=C(v)=1 otherwise
F(u,v) is the Discrete Cosine transformed 8*8 block

(ii) Quantization

The 8x8 block of transformed values is then divided by a distinct quantization value for each transformed block entry.
Fquantized(x,y) = F(x,y) / Quantization_Table(x,y)

(iii) Zigzag

Zigzag step takes the quantized 8x8 block and orders them in a 'zig-zag' sequence, resulting in a 1-D array of 64 entries, which is shown in Figure 6. This process helps entropy coding by placing low-frequency coefficients (usually larger values) before the high-frequency coefficients (usually close to zero).
As motion in the chair, the DCT is the most important and contains much of calculation. This part of the chain will be developed in SystemC, and represents part Hardware Figure5. We explain it using an example process named 'DCT' (in JPEG encoder) in SystemC as shown in Figure 6.

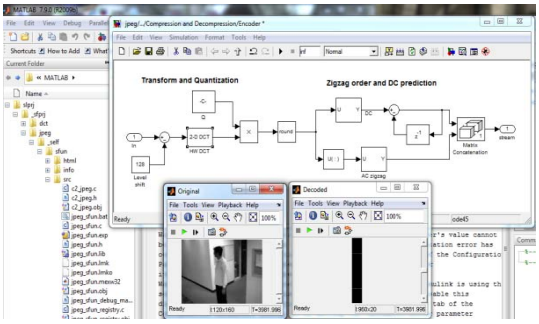

Figure 5: implementing the JPEG algorithm.

We explain it using an example process named 'DCT' (in JPEG encoder) in SystemC as shown in Figure 6.

```
struct fdct : sc_module {
sc_out<double> out64[8][8]; // the dc transformed 8x8 block
sc_in<double> fcosine[8][8]; // cosine table input
sc_in<FILE *> sc_input; // input file pointer port
sc_in<bool> clk; // clock signal
char input_data[8][8]; // the data read from the input file
void read_data( void ); // read the 8x8 block
void calculate_dct( void ); // perform dc transform
// define fdct as a constructor
SC_CTOR( fdct ) {
// read_data method sensitive to +ve & calculate_dct sensitive to
// -ve clock edge, entire read and dct will take one clock cycle
SC_METHOD( read_data ); // define read_data as a method
dont_initialize();
sensitive_pos << clk;
SC_METHOD( calculate_dct );
dont_initialize();
sensitive_neg << clk;
}
};
```

Figure 6: The DCT in systemC.

It has two FIFO channels, one for receiving data and the other for sending data. From the SystemC code, we remove all SystemC dependent statements and exchange the FIFO read/write.

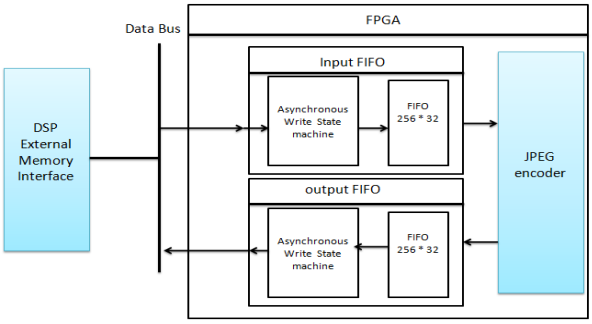

Figure 7: Two FIFO channels.

To proceed to an FPGA implementation, the resulting netlist from the previous stage has to be mapped to the FGPA's logic block structure and interconnect. The main outcome of this technology mapping, placing, and routing is a bit stream which can be programmed into a FPGA figure 7.

The virtual architecture is modeled using transaction level modeling (TLM) techniques that allow analyzing FPGA architecture in an earlier phase of design, software development and timing estimation. At the virtual architecture level, the Simulink functions of the application are transformed into systemC program code for each task. This step is very similar to the code generation performed by Real Time Workshop (RTW).Contrary to the RTW which generates only single task code, the software at the virtual architecture level represents a multitasking systemC code description of the initial Simulink application model.The development of the JPEG Decoder application in Simulink requires seven S-Functions in order to integrate the systemC code of the main parts of the decoding algorithm. Which are: jpeg_sfun_h, dct_sfun_h, sfc_sf.h, sfc_mex.h, sfcdebug.h, jpeg_sfun.mexw32, dct_sfun.mexw32.

Once this link is established, it opens up a wide range of additional capability to SystemC, like stimulus generation and data visualization. The first advantage of our technique is to use the right tool for the right task. Complex stimulus generation and signal processing visualization are carried out with MATLAB and Simulink while hardware verification is performed with SystemC verification standard. The second advantage is to have a SystemC centric approach allowing greater flexibility and configurability.
In this part, we make a comparison between the previous methodology based on the communication and the synchronization between both simulators

and the new approach which is based on the integration of systemC in matlab / Simulink in other applications.
CODIS is a tool which can automatically produces co-simulation instances for continuous/discrete systems simulation using SystemC and Simulink simulators.To evaluate the performances of simulation models generated in CODIS, they measured the overhead given by the simulation interfaces. The experiments have shown synchronization overhead of less than 30 % in simulation time [9]. In the [5] A Software-Defined Radio (SDR) is a combination of digital filters, analog components and processors, each requiring different design approaches with a different tool or language. Using a traditional design flow, where the verification effort represents 70% of the total design time, will yield in more time spent on testbench development and simulation runs. The result is 192 days as the total development time for this project, compared to 131 days using the improved design flow. This represents a productivity gain of around 32% over a traditional design flow that has limited testbench components reuse and software interoperability. In [10], we presented a check structure based on a new co-simulation interface between SystemC, MATLAB and the Simulink environment. This platform can be used to help the check of DSP cores design. By using MALTAB and Simulink environment, we improved the SystemC capability of material check. Bat we noticed that méthodologie is not enough. We can improve and integrated into systemC Matlab. The implementation HW/SW reduced the number of clock cycle: 1334722 to 158044 times of execution. The reduction on the total execution time of the JPEG algorithm was 88. 15%.

V. CONCLUSIONS

In this paper, we presented a new approach Based on the integration systemc in matlab / simulink. The capital advantage of this approach is the possibility of modeling and verifying the overall system within the same design environment. The result is shorter design cycles for applications using heterogeneous architectures. The co-simulation interface we presented a method for reducing the time spent on validation and verification while improving overall testbench quality. MATLAB/Simulink assists the SystemC verification environment in a unified approach. It has been shown that the methodology allows complex stimulus generation and exhaustive data analysis for the design under verification. As FPGA designs encompass larger and larger systems, the need to efficiently model the complex external environment during the architecture and verification phases becomes greater. The whole verification flow has been evaluated, using an example. It has been shown, that the usage of the extended verification flow saves a significant amount of time during the development process. The proposed plateform is tested on the JPEG compression algorithm. The execution time of such algorithm is improved by 88.15% due to the hardware implementation of the Matlab mult16 Function using SystemC. As future works, we aim to test our platform with the whole video compression chain using MPEG4 modules and Software-Defined Radio (SDR). It includes hardware and software components that require rigorous verification all along the design flow.

REFERENCES

[1] A. Avila, “Hardware/Software Implementation of a Discrete Cosine Transform Algorithm Using SystemC” Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2005)

[2] M.Abid, A. Changuel, A. Jerraya,” Exploration of Hardware/Software Design Space through a Codesign of Robot Arm Controller” EURO-DAC '96 with EURO-VHDL '96 pp 17-24

[3] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, M. Poncino, “SystemC Cosimulation and Emulation of Multiprocessor SoC designs,” Computer Magazine, April 2003 pp: 53 – 59

[4] The Open SystemC Initiative (OSCI) http://www.systemc.org

[5] J.F. Boland “Using MATLAB and Simulink in a SystemC Verification Environment”, Proc. of Design and Verification Conference & Exhibition, San Jose, Californie, Février 2005

[6] F. Czerner and J. Zellmann. “Modeling cycle-accurate hardware with matlab/ simulink using systemc”. 6th European SystemC Users Group Meeting (ESCUG), October 2002.

[7] C. Warwick. Systemc calls matlab. MATLAB Central, March 2003.

[8] The MathWorks. Link for ModelSim 2.0, 2006.

[9] F. Bouchhima, M. Briere, G. Nicolescu, M. Abid, and E.M. Aboulhamid. A SystemC/Simulink co-simulation framework for continuous/discrete-events simulation. In Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International, pages 1–6, 2006

[10]W.hassairi, M.Bousselmi, M.Abid,C.valderama “Using Matlab And Simulink In SystemC Verification Environment By JPEG Algorithm“ICECS 2009 ,page 912-915

[11] Draft Standard SystemC Language Reference Manual April 25 2005

[12] Independent JPEG Group, http://www.ijg.org

[13] Hiroyasu Mitsui “A Student Experiment Method for Learning the Basics of Embedded Software Development Including HW/SW Co-design” 22nd International Conference on Advanced Information Networking and Applications – Workshops 2008 pp.1367-1376

[14] Katalin Maria POPOVICI “Environnement de Programmation Multi Niveau pour Architectures Hétérogènes MPSoC ” these 2008 pp 90-92.