CO-SIMULATION INTERFACE SYSTEMC AND MATLAB VERIFICATION ENVIRONMENT BY 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

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 pressure to time-to-market has been continuously increasing in the past decades. Simulation of the entire system including both hardware and software from early design stages is one of effective approaches to improve the design productivity. A large number of research efforts on hardware/software co-simulation have been made so far. Real-time operating systems have become one of the important components in embedded systems. Therefore, in order to validate the entire system functionality, this system has to be simulated together with application software and hardware. 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. High-level test-benches, assertion-based verification, formal methods, hardware verification languages are just a few examples of the intense research activities driving the verification domain. A integrate SystemC in MATLAB/Simulink and is presented. This interface is principally used for the verification of lower abstraction level designs with a high level model of the design environment. Our work articulates on three contributions which are the proposal for solutions of implementation of different the parts of architecture in the case of the simulators: systemC and Simulink/matlab, moreover the definition of an environment of Co-simulation based on the automatic generation of the interfaces necessary for the integration these simulators and the proposed a new verification framework based on SystemC verification standard that uses MATLAB and Simulink to accelerate testbench development.

A verification methodology based on SystemC is proposed. The MATLAB/Simulink to SystemC interface and the evolved version of transactors are combined in a scalable multi-abstraction level verification platform.

This paper describes a refined co-simulation platform which overcomes these problems. The refined platform enables co-simulation with hardware models written in SystemC. On the co-simulation platform, all of the application software and hardware modules are directly executed on a host computer, which leads to high co-simulation speed. 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

ISBN: 978-9938-9511-0-3
International Conference on Information Technology and e-Services

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.

In this paper, we first discussed the related work in section 2 and in section 3, we presented a methodology of co-simulation. Then, in section 4, we proposed the application of compression image JPEG. After that, we give the resultant for co-simulation in section 5. Finally, we concluded by suggesting some 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] have look at the problem of cosimulating 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 approach 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], I 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 thus we can interact with Simulink solvers. For our bridge, I 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 1 picoseconds; this can be changed with function sc_set_time_resolution. Simulink maintains simulation time as a double precision value scaled to seconds. Our co-simulation interface uses a one-to-one correspondence between simulation time in Simulink and SystemC.

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.

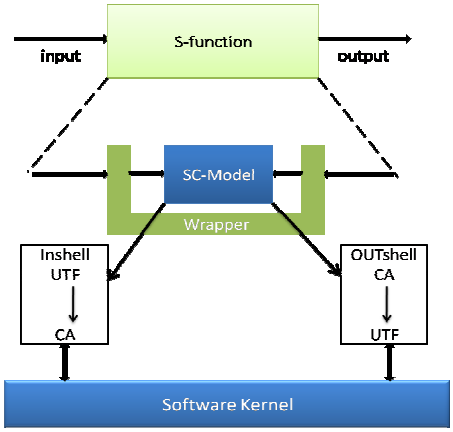

Fig. 1: Using SystemC within Simulink S-Function

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. Thus an integration of the visualized tool in more common design flows is realized.

This integration was done by using Simulink S-Functions. S-Functions provide a powerful mechanism for extending Simulink with custom blocks and can be implemented as C++ Code. Within the S-Function the output is calculated from input and from states at each time step when the fixed-step, discrete time solver is used.

So a stepwise, cycle by cycle SystemC-simulation is needed, which can be executed in the S-Function. The initialization of the SystemC kernel must be separated from simulation.

To meet these requirements a wrapper has been inserted between the S-Function and the SystemC model (Fig. 1). The purpose of that wrapper is:

- Connect Simulink ports to a SystemC-TM-Block.
- Converting Simulink data types to SystemC-TM signals and vice versa.
- Initializing of the SystemC-Kernel.
- Converting events; function call from Simulink to sc_cycle().
- Provide a DLL interface to the Simulink S-Function.

The 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 [10] 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.

With this approach the overall system simulation can be controlled by Simulink through settings of duration time and step size.

The machine vision operators designed and verified in Matlab/Simulink can be added to the hardware library of IPED and used as executable specification in the backend design process.

start_of_simulation: The implementation shall call member function **start_of_simulation** immediately the application calls function **sc_start** for the first time or at the very start of simulation if simulation is initiated under the direct control of the kernel. If an application makes multiple calls to **sc_start**, the implementation shall only make the callbacks to **start_of_simulation** on the first such call to **sc_start**. The implementation shall call function **start_of_simulation** after the callbacks to **end_of_elaboration** and before invoking the initialization phase of the scheduler.

end_of_simulation: The implementation shall call member function **end_of_simulation** at the point when the scheduler halts due to the function **sc_stop** having been called during simulation, or at the very end of simulation if simulation is initiated under the direct control of the kernel. The purpose of member function **end_of_simulation** is to allow an application to perform housekeeping actions at the end of simulation. Examples include closing stimulus and response files, and printing diagnostic messages. The intention is that an implementation that initiates elaboration and simulation under direct control of the kernel (in the absence of functions **sc_main** and **sc_start**) shall make the callbacks to **end_of_simulation** at the very end of simulation whether or not function **sc_stop** has been called.

IV. JPEG compression algorithm

The current standard for JPEG compression is a recommendation issued by ITU (International telecommunication Union) in 1992. JPEG is an image compression algorithm for continues-tones images and photographs. Several variants of the JPEG compression method exists. From research made on the human visual system, psycho visual models have been created. These models explain among other things what we are able to see and to what extent. These models, together with the FDCT

(FORWARD Discrete Cosine Transform), from the foundation of JPEG encoding.
When JPEG compression is applied on continuous-tone images, a high quality output can often be achieved with a compression ratio as high as 1:10 - 1:20. At a compression ratio of 1:5, the output is virtually indistinguishable from the original source.
The JPEG encoding process can be divided into five steps:

- Color conversion
- Sub sampling
- Two-dimensional FDCT
- Quantization
- Entropy encoding

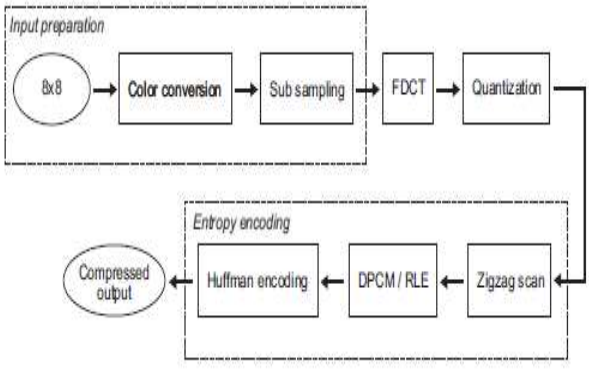

Fig. 2: The JPEG decoder

The image consists of one or several color components or color channels. The individual data elements the source image and the output image consist of are referred to as samples and coefficients.
Prior to the JPEG compression is started, the input image is divided into square data blocks of eight by eight samples. Data blocks representing each of the available color channels are grouped into Minimum Coded Units (MCU). A MCU represents the smallest possible subset of the encoded image. The size and arrangement of the MCU varies depending on which color format is used.
The studies made on the human visual system showed that humans are more sensitive to changes and details in brightness than in color. JPEG uses a color space that takes advantage of this fact, CIELAB or YUV. YUV consists of three components: Luminance (Y), Chrominance A (U) and Chrominance B (V). Luminance represents brightness, while the two chrominance channels together represents color.
By applying a 2D FDCT on the source image, it is transformed from the spatial domain to the frequency domain. This transformation basically separates visually important image detail from detail of less visual importance (figure 2.14).
For each data block of YUV samples from the source image, a corresponding data block of coefficients is created as a result of the 2D FDCT. Together all coefficients render their data block. Each coefficient adds image detail to the data block. The coefficients are ordered according to their significance to image detail, from the top-left corner diagonally downwards to the bottom-right corner.
Since the 2D FDCT divided image detail information according to their importance, this knowledge makes it possible to further remove redundant data. Quantization vectors are utilized to individually scale each coefficient after their significance and after the sought compression ratio. The ITU recommendation contains suggested quantization tables that are carefully balanced between image quality and compression ratio.
After quantization, each data block is zigzag scanned as a preparation for entropy encoding. The zigzag scan reorganizes the coefficients in an order which makes the entropy encoding more efficient in each data block, the upper, leftmost coefficient represents an average value for the entire data block. It is referred to as the DC coefficient while the other 63 coefficients are referred to as the AC coefficients.
The DC value is encoded with Differential Pulse Code Modulation (DPCM). DPCM enhances the compression performance for continuous-tone images by encoding the difference between adjacent data blocks' DC coefficients.
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. We explain it using an example process named 'DCT' (in JPEG encoder) in SystemC as shown in Figure 3.

```
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;
  }
};
```

Fig. 3: 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.
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.

V. Results

The virtual architecture model is described using SystemC language and is generated according to the parameters specified in the initial Simulink model. SystemC allows modeling a system at different abstraction levels from functional to pin accurate register transfer level.
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 generation has to support also user defined systemC codes integrated in the Simulink model as S-functions. For the S-functions, the task code represents a function call of the user written systemC function. The semantics of the argument passing are identical to those of the definition in the configuration panel of the S-Function Builder tool in Simulink. The hardware is refined to a set of abstract SystemC modules (SC_MODULE) for each subsystem. The SC_MODULE of the processor includes the tasks modules that are mapped on the processor and the communication channels for the intra-subsystem communication between the tasks inside the same processor. The communication channels between the tasks mapped on the FPGA is implemented using standard SystemC channels. The tasks modules are implemented as SystemC modules (SC_MODULE). The development of the JPEG Decoder application in Simulink requires 7 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 (COntinuous DIscrete Simulation) is a tool which can automatically produces co-simulation instances for continuous/discrete systems simulation using SystemC and Simulink simulators. This is done by generating and providing co-simulation interfaces and the co-simulation bus. 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. Bat 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%.

VI. 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