International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.2, April 2012

A SYSTEMC/SIMULINK CO-SIMULATION ENVIRONMENT OF THE JPEG ALGORITHM

Walid Hassairi, Moncef Bousselmi, Mohamed Abid and Carlos Valderrama

UMons University of Mons, Electronics & Microelectronics Dpt., Mons, Belgium
Laboratory CES, National School of Engineers of Sfax, Tunisia

1. INTRODUCTION

In the past decades, many factors have been continuously increasing like the functionality of embedded systems as well as the time-to-market pressure has been continuously increasing. 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 the embedded systems. However, in order to validate function of the entire system, 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 too slow to execute. New methods and verification techniques began to emerge over the past few years. Highlevel test-benches, assertion-based verification, formal methods, hardware verification languages are just a few examples of the intense research activities driving the verification domain.

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. Second the definition of a co-simulation environment based on the automatic generation of the interfaces required to the integration of these simulators. Finally the proposal of a new verification framework based on SystemC Verification standard that uses MATLAB/Simulink to accelerate the test-bench 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

DOI : 10.5121/vlsic.2012.3201

Encoding), the SW part implemented in Matlab. With this research premise, this study introduces a new HW implementation of the DCT algorithm in SystemC. For the communication and synchronization between these two parts we use the S-Function and the MATLAB/Simulink engine. In addition, we compare the co-simulation results to a pure software simulation.

In this chapter, the related work is discussed in Section 2 and the proposed co-simulation methodology is presented in Section 3. Then, in Section 4, we propose the implementation of the JPEG image compression as a case study. In Section 5, we summarize the proposed approach and co-simulation results. Finally, we sum up the proposal including suggestions and recommendations to future works.

2. RELATED WORK

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

This wrapper initializes the SystemC kernel and converts Simulink data to SystemC signals. Simulation control is entirely handled by Simulink. 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 work demo shows how to use the library to send and retrieve data from the MATLAB workspace. 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 the 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 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 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 work [10], they have created a tool called: co-simulation COLIF that defines a subset of Matlab / simulink and combines a set of descriptive rules allows for the specification and

functional validation efficient algorithms for the application. To reduce the "gap" between the functional model and architecture model in SystemC, they proposed a new intermediate transactional model in Simulink executable that combines both the algorithm and architecture in a single model representation. To validate their work, they applied to decoder MPEG Layer III. They found that the simulation model in Simulink is 50 times faster than the macro-level architecture. The difference is mainly due to the complexity of the description and details of the communication are present at the macro architecture.

In our former work [11], we adopted the methodology of communication and synchronization. To exchange data between a Simulink model and SystemC module, the cosimulation interface must integrate a bridge between the two simulators. This bridge is built with two Simulink S-Functions. An S-Function is a 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 picosecond. This limit can be changed with the function sc_set_time_resolution. However, the time in Simulink simulation is a double precision value scaled to seconds. Thus, our co-simulation interface uses a one-toone correspondence between simulation time in Simulink and SystemC.

3. METHODOLOGIES

The implementation of applications on embedded systems is a very time expensive task using the standard development tools. The proposed heterogeneous model is also executable to simulate the co-design implementation. Such simulation of the heterogeneous model is realized using SystemC. In fact, a description of a hardware module is transformed into a structural description with SystemC components (RT-level). Then, the interface between hardware and software parts is implemented using special SystemC constructs. This interface 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 given by figure Fig1, 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.

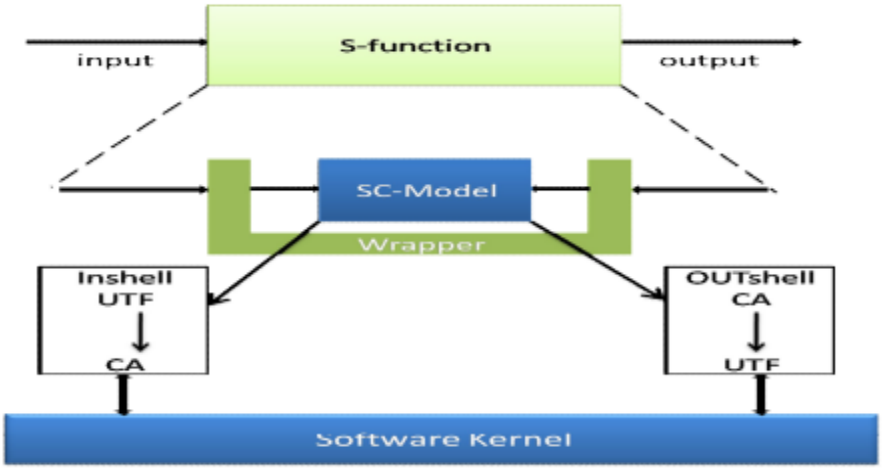

Fig. 1: Integrated SystemC in Simulink S-Function.

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, the visualized tool in more common design flows is integrated using Simulink S-Functions. Those 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 using a cycle by cycle SystemC-simulation as a fixedstep discrete time solver. The initialization of the SystemC kernel should be separated from simulation.

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

- connecting 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(),
- providing a DLL interface to the Simulink S-Function.

So, our methodology tries to pull 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. SystemC generate stimulus and data visualization [10]. We also based our methodology on a portion of the methodology in the work [11]. In this work, they are based on the transformation of a task in SystemC. The first benefit 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 benefit is to have a SystemC centric approach allowing greater configurability and flexibility.

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

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. The novel methods are:

void before_end_of_elaboration();

This method is called just before the end of elaboration processing is to be done by the simulator.

void start_of_simulation();

This method is called just before the start of simulation. It is intended to allow users to set up variable traces and other verification functions that should be done at the start of simulation.

void end_of_simulation();

If a call to sc_stop() had been made this method will be called as part of the clean up process as the simulation ends. It is intended to allow users to perform final outputs, close files, storage, etc.

It is also possible to test whether the callbacks to the start_of_simulation methods or end_of_simulation methods have occurred. The Boolean functions sc_start_of_simulation_invoked() and sc_end_of_simulation_invoked() will return true if their respective callbacks have occurred.

The tasks at the transactional level under Simulink are included in a software knot represented by a sub-system having the prefix ' SW_ ' in its name. These tasks are modeled under Simulink in several ways.

They can be trained by a merger of several blocks in one under system having the name preceded by the prefix ' TASK _ ' either they are trained by individual blocks. These last ones, in turn can be predefined blocks of the library either Functions modelled in language C.

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.

For the modelling and description 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 behaviour using the elementary modules "SC_CTHREAD".
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. 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.

```
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())
};
};
```

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

However, the figure 3 shows the main file. "cpp '. The main calculation is done to the body of this task. The communication of this module with the system is through the interfaces represented by the ports of entry and exit 'DATA_IN1'and 'DATA_OUT1' by means of APIs defined in the library.

```
#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);
          };
}
```

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

3.1 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 conception process always begins with the specification of the application in the Simulink environment using S-Functions blocks. 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 the implementation file in C + +. To understand better the transformation of one S-Function into a task, we divided into four parts.

In the first part, 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.

In the second part, 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).

In the third part, the APIs and the 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.

In the final part, 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.

3.2 Creating a task from a SystemC predefined block in the Simulink library.

In the case of an elementary block a different type of S-function included in a software node (a subsystem with the prefix 'SW_'), the generation of the tasks SystemC is made from a bookshop of functions describing the behaviour of all the blocks Simulink used in the application.

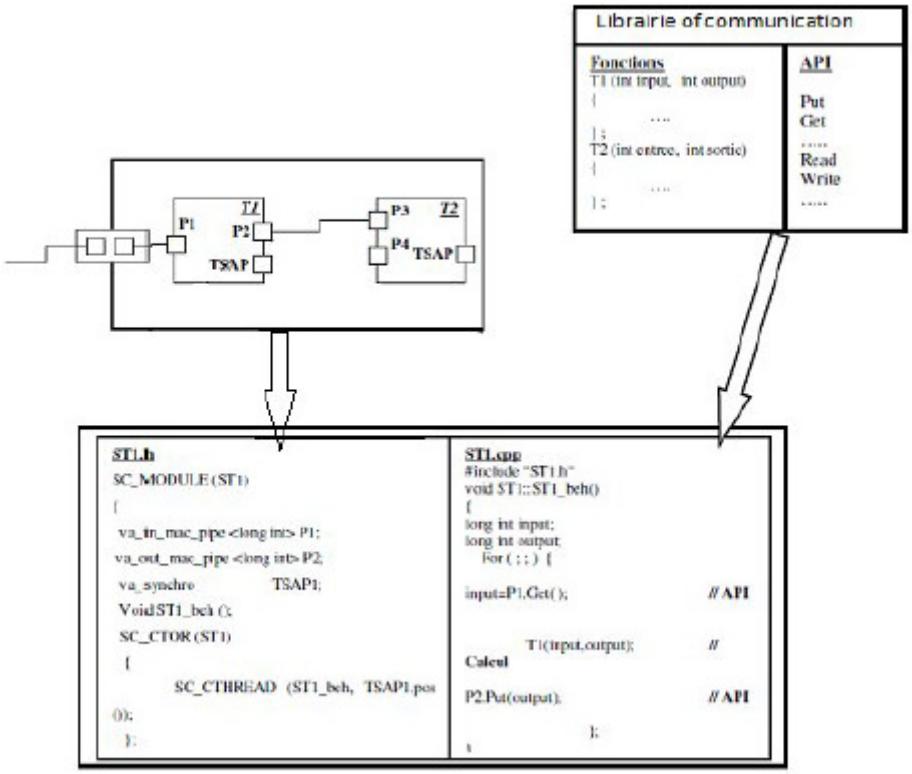

Fig 4: Generating a task from a basic block.

Each function has the same name as the Simulink block and the corresponding module in our methodology. However, reading and writing data are specific through the APIs to each communication protocol. These APIs exist in the communication library. The type of communication protocol is identified in the 'Port' of each module in our methodology. Figure 4 shows the generation of a task in SystemC from an individual block in Simulink transaction, this block is transformed into a parameterized module under our methodology.

3.4 Fusion of several blocks Simulink in one task SystemC.

In the case where several units are grouped in a subsystem representing a task whose name is prefixed with 'TASK_' the generation of the task SystemC is by assembling several library functions into a single task SystemC. Functions have the same names of the blocks. These functions exchange data via common variables. Communication with the system 'inter_Thread' is via the APIs generated following the protocol communication defined in our methodology.

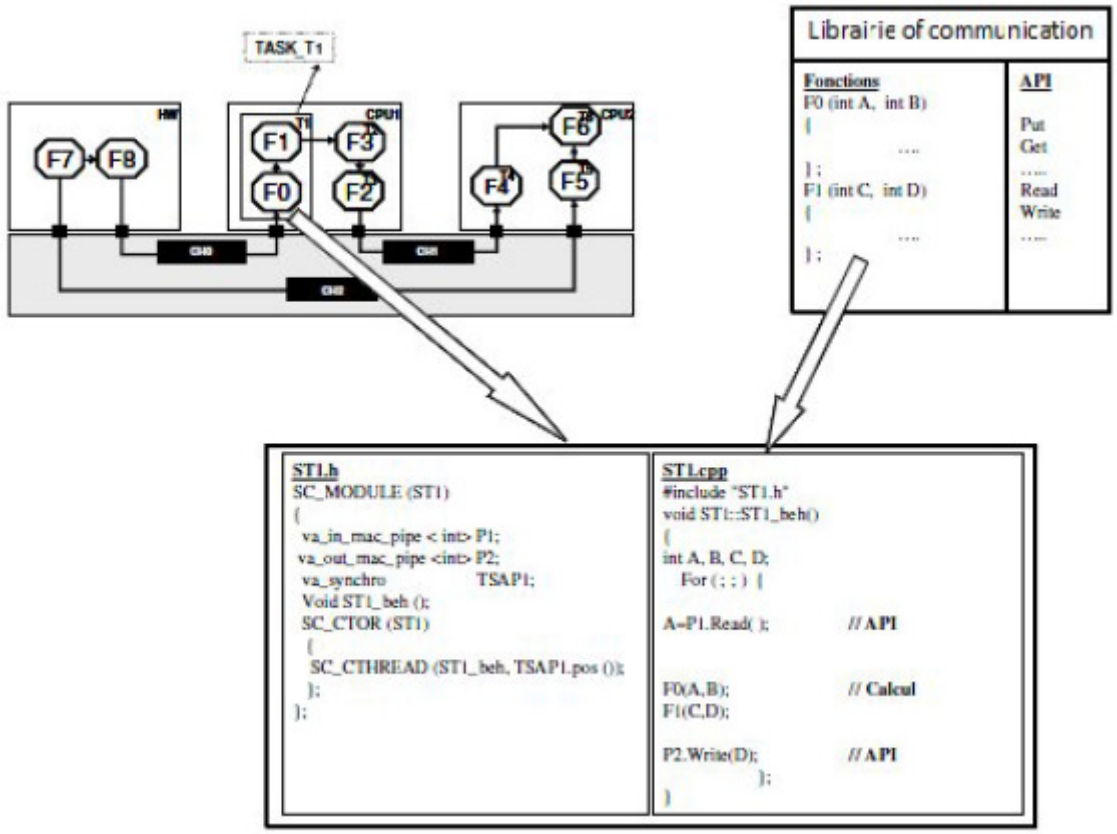

Fig 5: Generating a task from a set of blocks in Simulink

Figure 5 illustrates the merger of several blocks in Simulink transactional to generate a task in SystemC. The functions of the library F0 (), F1 () have the same names as the blocks F0, F1. The generation of APIs is done by identifying the type of protocol in each port of the module in the virtual architecture of our methodology.

4. JPEG COMPRESSION ALGORITHM

The baseline JPEG compression algorithm is the most basic form of sequential DCT based compression [12]. The process of JPEG-based encoding and decoding of images vary according to color depth (8, 24 or 32 bits). However, the basic ideology for all color depths is same. The bitmap image stores raw pixel-by-pixel color values. In addition, 54 bytes are stored at the start of file as header information that includes image width and height, image file size, image color depth, etc. These 54 bytes must be taken into account whenever working with the bitmap images. Following the 54-byte header, the bitmap image holds the color values of each pixel that varies for different color depths. 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. For 8-bit pixels, the pre-processing stage divides image data into 8x8 blocks that are shifted from unsigned integers with range $[0, 2^8 - 1]$ to signed integers with a range of $[-2^7, 2^7 - 1]$ and then individually compressed at the 8x8 block level. 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.

4.1 FDCT and IDCT 8x8

The input to the encoder, source image samples are grouped into 8x8 blocks, shifted from unsigned integers with range [0, 2^7 - 1] to signed integers with range [-2^7-1, 2^{7-1}-1], and input to the Forward DCT (FDCT). The output from the decoder, the Inverse DCT (IDCT) outputs 8x8 sample blocks to form the reconstructed image. The following equations are the idealized mathematical definitions of the 8x8 FDCT and 8x8 IDCT:

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

$$x, y = 0,1 \ldots 7$$

$$c(u,v) = \begin{cases} \frac{1}{2} where, u = v = 0 \\ \frac{1}{\sqrt{2}} where, u = 0, v \neq 0 \\ \frac{1}{\sqrt{2}} where, u \neq 0, v = 0 \\ 1, atherwise \end{cases}$$

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

The DCT is related to the Discrete Fourier Transform (DFT). Some intuition for DCT-based compression can be result by viewing the FDCT as a harmonic analyzer and the IDCT as a harmonic synthesizer. All 8x8 block of source image samples is effectively a 64-point discrete signal which is a function of the two spatial dimensions x and y. The FDCT takes such a signal as its input and decomposes it into 64 orthogonal basis signals. All contains one of the 64 unique two-dimensional (2D) "spatial frequencies'' which comprise the input signal's "spectrum." The output of the FDCT is the set of 64 basis-signal amplitudes or "DCT coefficients" whose values are uniquely determined by the particular 64-point input signal.

The DCT coefficient values can thus be regarded as the relative amount of the 2D spatial frequencies contained in the 64-point input signal. The coefficient with zero frequency in both

dimensions is called the "DC coefficient" and the remaining 63 coefficients are called the "AC coefficients.'' Because sample values typically vary slowly from point to point across an image, the FDCT processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. For a typical 8x8 sample block from a typical source image, most of the spatial frequencies have zero or near-zero amplitude and need not be encoded.

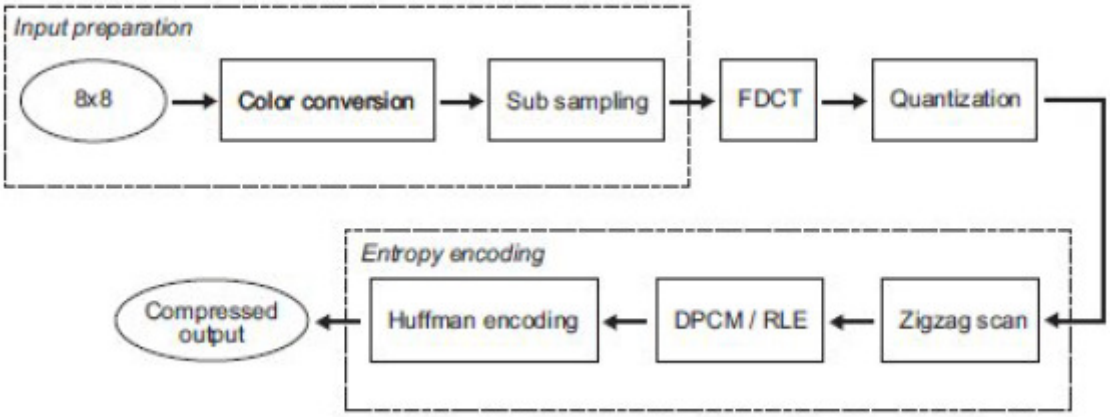

Fig 6: The JPEG decoder.

For the IDCT reverses, it takes the 64 DCT coefficients and reconstructs a 64-point output image signal by summing the basis signals. If the FDCT and IDCT could be computed with perfect accuracy and if the DCT coefficients were not quantized as in the following description, the original 64-point signal could be exactly recovered. In principle, the DCT introduces no loss to the source image samples. It just transforms them to a domain in which they can be more efficiently encoded. Some properties of practical FDCT and IDCT implementations raise the issue of what precisely should be required by the JPEG standard. A fundamental property is that the FDCT and IDCT equations contain transcendental functions.

4.2 Quantization

After output from the FDCT, each of the 64 DCT coefficients is consistently quantized in conjunction with a 64-element Quantization Table, which must be specified by the application (or user) as an input to the encoder. Each element can be any integer value from 1 to 255, which specifies the step size of the quantizer for its corresponding DCT coefficient. The goal of this processing step is to discard information which is not visually significant. Quantization is a many-to-one mapping, and therefore is fundamentally lossy. It is the principal source of lossiness in DCT-based encoders. Quantization is defined as division of each DCT coefficient by its corresponding quantizer step size, followed by rounding to the nearest integer:

$$F^{Q}(u,v) = Integer\ Round\ (\frac{F(u,v)}{Q(u,v)}) \qquad (3)$$

This output value is unified by the quantizer step size. Dequantization is the inverse function, simply means in this case that the normalization is removed by multiplying by the step size, which returns the result to a representation appropriate for input to the IDCT:

$$F^{Q'}(u,v) = F^{Q}(u,v) * Q(u,v) \quad (4)$$

When the aim is to compress the image as much as possible without visible artifacts, each step size ideally should be chosen as the perceptual threshold or "just noticeable difference" for the visual contribution of its corresponding cosine basis function. These thresholds are likewise functions of the source image characteristics, display characteristics and viewing distance. For applications in which these variables can be reasonably well defined, psycho visual experiments can be performed to determine the best thresholds.

4.3 DC Coding and Zig-Zag Sequence

After quantization, the DC coefficient is treated in isolation from the 63 AC coefficients. The DC coefficient is a measure of the average value of the 64 image samples. Because there is usually strong correlation between the DC coefficients of adjacent 8x8 blocks, the quantized DC coefficient is encoded as the difference from the DC term of the previous block in the encoding order (defined in the following), as shown in Figure 7. This special treatment is worthwhile, as DC coefficients generally contain a significant fraction of the total image energy.

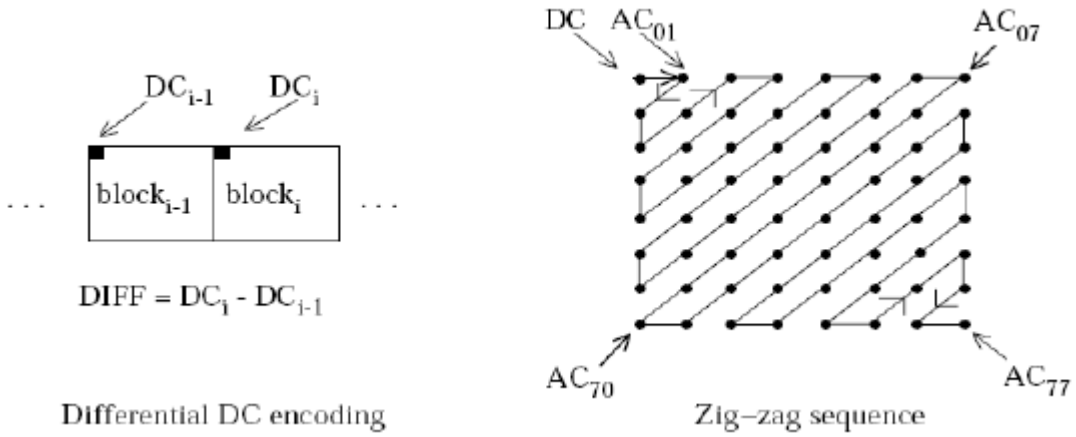

Fig 7: Preparation of Quantized Coefficients for Entropy Coding

Finally, all of the quantized coefficients are ordered into the "zig-zag" sequence, also shown in Figure 7. This ordering helps to facilitate entropy coding by placing low-frequency coefficients (which are more likely to be nonzero) before high-frequency coefficients.

4.4 Entropy Coding\Huffman

Huffman coding is a technique which will assign a variable length codeword to an input data item. Huffman coding assigns a smaller codeword to an input that occurs more frequently. It is very similar to Morse code, which assigned smaller pulse combinations to letters that occurred

more frequently. Huffman coding is variable length coding, where characters are not coded to a fixed number of bits.

This is the last step in the encoding process. It organizes the data stream into a smaller number of output data packets by assigning unique code words that later during decompression can be reconstructed without loss. For the JPEG process, each combination of run length and size category, from the run length coder, are assigned a Huffman codeword.

4.5 Decomposition and implementation of the JPEG algorithm

It is possible to increase speed and to reduce power consumption by running portions of the algorithm implemented in the custom hardware. To do this, parts of the algorithm remains the SW and the other part goes to HW area and must be well chosen. This is called hardware partitioning software (HW / SW partitioning). Many factors must be considered when the HW / SW partitioning is done. The problem is to utilize the right amount of material. To use too much material implies a rise in costs and probably increase the time of placing on the market.

The first step in a HW / SW partitioning is to identify the parts of the algorithm that consumes a lot of time if left in the software or by the implementation of the algorithm entirely in software or perform estimates on the number of cycles. The next step is to evaluate and decide which parts need to be moved to the HW area. It is important to take into account more things than just a party that consumes more cycles of the software. Perhaps it is better to leave this part of computation in software intensive and move some other parts in HW, the parts that are better suited for hardware implementation. This is of course possible only if time constraints may even now be suffering the most intense in the software calculation.

To make a good HW / SW partitioning a simulation tool is needed where much can be moved from HW field to SW field and vice versa. In addition, it should be possible to specify the execution time for different parts. This part of the design process is important and time spent here is well spent and often reduces the work in phases. If the processor architecture likewise must be chosen in the design process, the problem becomes even more complicated. With a more powerful processor, it is probably possible to do more in software and thus reduce the cost of designing and manufacturing the hardware. The question then is of course how this affects the total cost. The entire HW / SW partitioning problem is an optimization problem where constraints are typical on the surface of silicon, energy, monetary cost and execution time. So the time aspect of the market must be considered. In this section, we illustrate the approach we have followed for the implementation of JPEG through our methodology. As we have previously presented the most important part of the chain compression and DCT part, it has a lot of calculating. In this case we will implement this part with SystemC and the rest of the chain compression is implemented on MATLAB. Figure 8 shows the implementation of the encoder jpeg presenting all the different parties.

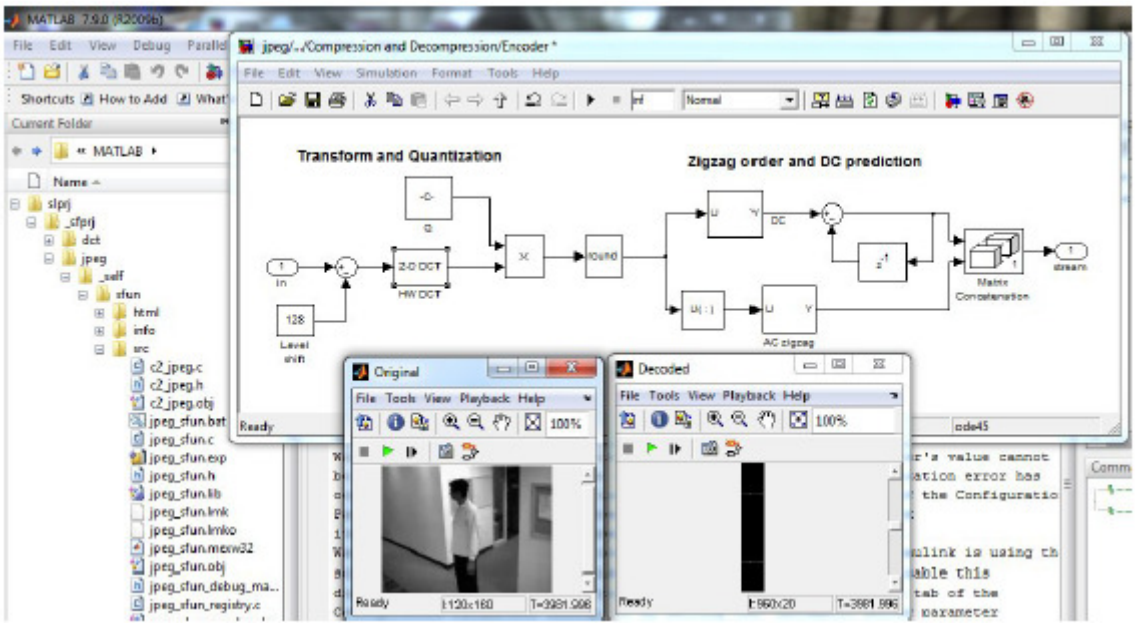

Fig 8: implementing the JPEG algorithm.

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 Figure8. We explain it using an example process named 'DCT' (in JPEG encoder) in SystemC as shown in Figure 9.

```
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 9: 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.

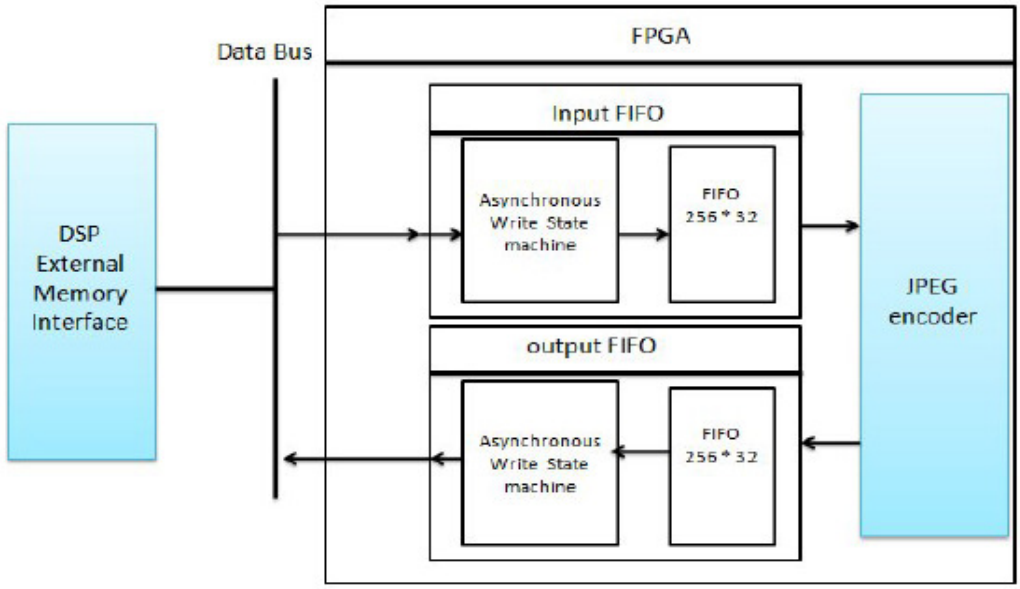

Fig 10: Two FIFO channels.

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 10.

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.

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 SFunction 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 gain 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 gain 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 cosimulation 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 test-bench 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 test-bench components reuse and software interoperability. But 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%.

5. Conclusion

In this chapter, 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 test-bench 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 platform 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] Youssef ATAT "Conception de haut niveau des MPSoCs à partir d'une spécification Simulink : Passerelle entre la conception au niveau Système et la génération d'architecture"21 Mai 2007

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

[12] Draft Standard SystemC Language Reference Manual April 25 2005 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] James Rosenthal " JPEG Image Compression using an FPGA" A Thesis submitted in partial satisfaction of the requirements for the degree Master of Science in Electrical and Computer Engineering December 2006