SystemC Co-Design for Image Compression: Fast Discrete Cosine Transform using Distributed Arithmetic Method

Mildred C. Zabawa, Malek Adjouadi, and Naphtali Rishe
Department of Electrical & Computer Engineering, Florida International University, 10555 W. Flagler Street, Miami FL 33174
http://www.cate.fiu.edu

Abstract
In this highly-technical, industrial driven market, companies are striving to sustain their competitive edge by researching and implementing methodologies which reduce development time and introduce product-lines to consumers quickly and cost effectively. From a Hardware Architect's perspective, designing real-time hardware involves various stages of modeling. A new, emerging hardware description language (HDL) called SystemC is currently capturing industries attention since it introduces a broad range of abstract level model techniques under one platform. SystemC is also an open-source language. Consumers are capable of stimulating various levels of modeling from the transaction level (TL) to the Register Transfer Level (RTL). Companies are investigating these potential strategic changes of functional verification stages in order to compress development time. In order to meet future demand, Hardware EDA Vendors such as Synopsys and Mentor Graphics have introduced new EDA tools that support SystemC synthesis. This introduces a new paradigm for future change of the support and functional verification stages of product development. In today's market, co-hardware/software systems are being developed to determine the best-fit location of these modules. With this research premise, this study introduces a new implementation of a Hardware SystemC Register Transfer Level Model of a 2D-Discrete Cosine Transform using Distributed Arithmetic Algorithm. This model is integrated into an image and video compression system.

***Key-words*:** Discrete Cosine Transform, SystemC, Distributed Arithmetic Algorithm

1. Introduction

Semiconductor companies are striving to reduce cost by shortening functional hardware verification development time for complex integrated circuits. SystemC is one of the leading trends. SystemC is an extension C++ class library designed to aid in hardware modeling. Some of the key features of SystemC is its ability to model concurrent processes, timed events, and cycle-accurate events. SystemC offers the system engineer the opportunity to model hardware digital systems with various levels of abstraction. Mixed modes of abstraction levels can be incorporated into the target hardware's SystemC model. Some of the levels of modeling are un-timed event, transaction level, timed event, and register transfer level modeling. *Un-timed events modeling* is a style of hardware modeling not depending on time events such as clock cycles. This style of modeling is usually performed during feasibility and performance studies of the digital system. *Transaction Level* modeling is a style of hardware modeling dependent on transaction accuracies of events. System Transaction Level modeling are designed in this manner to model the transaction of how individual modules interact with one another at the interface levels. *Timed events* are functional models of hardware modules that are dependent on time events. *Register Transfer Level* modeling is a style of modeling concurrent process, timing events and interface input/output precisions of hardware digital systems under design consideration. After full-regression testing has been performed on the various levels of the target hardware model, it becomes possible to proceed with synthesizing gates. SystemC Modeling offers the system engineer the potential to develop co-hardware/software designs from transaction leveling to register transfer level of the stage of the development cycle and the verification approaches designed to meet emerging changes for digital system design. SystemC Modeling is being integrated in semiconductor design industries to provide immediate value for simulating real-time digital

systems. Modeling designs of digital systems can be broken into individual modules of various abstract levels. As individual modules become more accurate these individual modules can be refined into register transfer level models [1, 2].

2. Image/Video Compression

In the embedded systems conference of San Francisco, a performance estimation approach of MPEG 4 algorithms on ARM based designs using co-verification was introduced. In 2000, a study by Ming-Hau Lee [3] designed and implemented a high-performance low-power MorphoSys Reconfigurable Processor for data-parallel applications in the area of video compression. In 2003, the European IST-2001-34410 Camellia project involved the design of image core applications for smart cameras applications which integrate the application with video compression architecture. As the years progress, it is anticipated that the number of image and video compression projects linked to wireless multimedia applications would increase significantly. The overall goal of all these projects is the design of efficient compression architectures. Now, a new era is being introduced in the design of co-simulation environments using SystemC. Dan Crisu, [4] introduced a hardware/software co-simulation environment for graphics accelerator development in ARM-based SOCS. Sayinta et al.[5] introduce a mixed abstraction level co-simulation case study using SystemC for System on Chip Verification. Other research has been performed by Microsoft to incorporate SystemC with multimedia applications. The image/video compression system integrates the above concepts to introduce a new product line towards functional verification flow for SystemC Image/Video Compression Architectures.

3. Co-design System

The proposed design consists of a SystemC Image Encoder/Decoder whose architecture is shown in Figure 1.

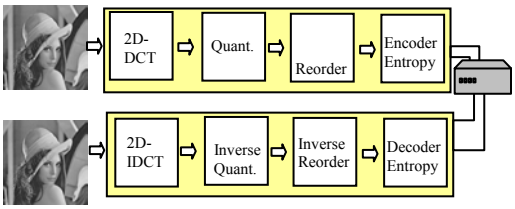

Figure 1: Image Encoder/Decoder System

In this architecture, a still image or a Planar 420 Video Frame is broken into 8x8 macroblocks. These individual macroblocks are then processed to be encoded in the following order: *2D-DCT* – converts spatial data into transform coefficients in the transform domain. *Quantizer* – Reduces coefficients into a domain representing strength in image importance. *Reorder* – reorders the 8x8 quantized coefficients into a single array form in a zig-zag pattern. *Encoder Entropy* – encodes using Huffman Entropy Encoding process based on run-length data and DC/AC look-up tables coefficients. *2D-IDCT* – converts transform coefficients into spatial data coefficients. *Inverse Quantizer* – Rescales coefficients into DCT coefficients. *Inverse Reorder* – reorders the single array to 8x8 quantized coefficients. *Decoder Entropy* – decodes using Huffman Entropy Decoding process based on run-length data and DC/AC look-up tables coefficients. The image encoder (IE) and image decoder (ID) will be used as modules for our video encoder/decoder system as shown in Figure 2:

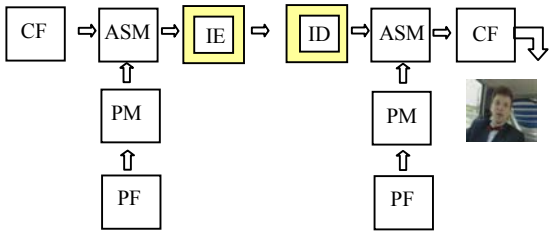

Figure 2: Video Encoder/Decoder System

In reference to Figure 2, the current frame (CF) and previous frame (PF) are processed through the Prediction Module(PM). The PM outputs are sent to the Adder/Subtractor (AS) Module. The AS module adds or subtracts the given input frames. The output result from the AS module is sent into the image encoder system. Likewise, a similar process is done to decode the video frame. Within the scope of this article, we will only focus on the architecture of the 2D-DCT component. Future articles will describe in detail the overall codec (encoder/decoder) systems.

4. 2D-DCT SystemC Architecture

The *2D Forward Discrete Cosine Transform (DCT)* is one of the major components in image and video compression architectures. It has proven to be one of the most efficient compression algorithms in today's multimedia

applications such as JPEG, MPEG1, MPEG2, and MPEG4. It converts spatial data into transform coefficients. This transform is reversible, symmetric, and orthogonal. These key components aid in its efficient implementation in co hardware/software systems. This transform is designed to de-correlate the transform coefficients into various level of frequencies where the 'peak' energy area (low frequencies) is located in the top-left corner. The strength in energy is decreased at its high-frequency coefficients. The DCT transform coefficients can be de-correlated where the smaller peak of energy areas (high frequencies) can be discarded without impacting the quality of the image being reconstructed. In the proposed SystemC implementation, design constraints had to be imposed on the image height and width directions to be module of 8 thus making it capable of being broken down into 8x8 macroblocks. In our case scenario, we will discuss the process for a grayscale still image, "lena", that is 256 x 256. This image is broken down to 32 macroblocks in each direction. This is depicted in Figure 3:

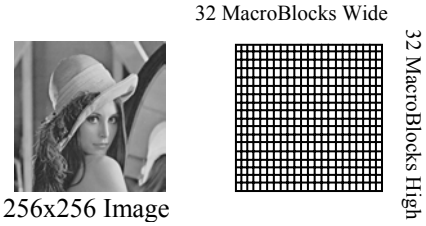

Figure 3: Lenna Grayscale Image 256x256 Image broken into 32x32 Macroblock

Each individual macroblock is 8x8 in dimensions. The 2-D Forward DCT Transform (FDCT) is performed on each 8x8 macroblock. The 2D-Forward Discrete Cosine Transform is based on the following well-known formula:

$$F_{x,y} = \frac{C(x)C(y)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} f_{i,j}\cos\left(\frac{(2i+1)x\pi}{16}\right)\cos\left(\frac{(2j+1)y\pi}{16}\right) \quad (1)$$

Where $C(n) = \begin{cases} \frac{1}{\sqrt{2}}, & n = 0 \\ 1, & n \neq 0 \end{cases}$

In order to perform an efficient hardware implementation using SystemC, the above is rewritten into separable 1-D transforms where $F_{x,y}$ are the DCT coefficients:

$$F_{x,y} = \frac{C(y)}{2}\sum_{j=0}^{7} F_x \cos\left(\frac{(2j+1)y\pi}{16}\right) \quad (2)$$

Where $F_x = \frac{C(x)}{2}\sum_{i=0}^{7} f_i \cos\left(\frac{(2i+1)x\pi}{16}\right)$

Thus, F_x (i = 0, 1, …,7) are the eight coefficients, and f_i are the input samples for the corresponding 1x8 row in the 8x8 macroblock. Thus, the separability property as used in this equation is one of highly desirable features of the DCT since it enables us to implement the real-time hardware architecture for a 2D-DCT in a more efficient manner. Therefore, two 1D-DCT algorithms are used for hardware simplicity. This enables us to implement the individual 1D-DCT algorithms in a distributed arithmetic approach.

5. Distributed Arithmetic: 1D-DCT

The basis of the Distributed Arithmetic Algorithm is to manipulate for the design under test mathematical equations to use only logical arithmetic operations. In our case, we re-formulate the 2D-DCT equations into two individual 1D Forward Discrete Cosine Transforms based on separability. Therefore, we are able to write this 1D-DCT as

$$F_x = \sum_{i=0}^{7} C_{i,x} f_i \quad where \quad C_{i,x} = \frac{C(x)}{2}\cos\left(\frac{(2i+1)x\pi}{16}\right) \quad (3)$$

The following Table 1 illustrates the corresponding coefficients for $C_{i,x}$ where A = 0.353553, B=0.490393, C = 0.415735, D = 0.277785, E = 0.097545, F = 0.461940, and G = 0.191342.

Table 1: $C_{i,x}$ Coefficients

	i=0	i=1	i=2	i=3	I=4	i=5	I=6	I=7
$C_{i,0}$	A	A	A	A	A	A	A	A
$C_{i,1}$	B	C	D	E	-E	-D	-C	-B
$C_{i,2}$	F	G	-G	-F	-F	-G	G	F
$C_{i,3}$	C	-E	-B	-D	D	B	E	-C
$C_{i,4}$	A	-A	-A	A	A	-A	-A	A
$C_{i,5}$	D	-B	E	C	-C	-E	B	-D
$C_{i,6}$	G	-F	F	-G	-G	F	-F	G
$C_{i,7}$	E	-D	C	-B	B	-C	D	-E

Note the symmetry property of the DCT. Equation 3 we can thus expand it as follows:

$$F_x = f_0C_{0x} + f_1C_{1x} + f_2C_{2x} + f_3C_{3x} + f_4C_{4x} + f_5C_{5x} + f_6C_{6x} + f_7C_{7x} \quad (4)$$

With the results in Table 1 and the expanded equation above the following symmetry can be revealed:

$C_{0,x} = C_{7,x}$ $C_{1,x} = C_{6,x}$ $C_{2,x} = C_{5,x}$ $C_{3,x} = C_{4,x}$ when x = 0, 2, 4, 6 and $C_{0,x} = -C_{7,x}$ $C_{1,x} = -C_{6,x}$ $C_{2,x} = -C_{5,x}$ $C_{3,x} = -C_{4,x}$ when x = 1, 3, 5, 7

Therefore, Equation 4 can be rewritten for even symmetry of x ($x=0,2,4,6$) as:

$$F_x = (f_0 + f_7)C_{o,x} + (f_1 + f_6)C_{1,x} + (f_2 + f_5)C_{2,x} + (f_3 + f_4)C_{3,x} \quad (5)$$

yielding:

$$F_x = u_0C_{o,x} + u_1C_{1,x} + u_2C_{2,x} + u_3C_{3,x} \quad (6)$$

Similarly for odd symmetry of x ($x=1,3,5,7$) :

$$F_x = (f_0 - f_7)C_{o,x} + (f_1 - f_6)C_{1,x} + (f_2 - f_5)C_{2,x} + (f_3 - f_4)C_{3,x} \quad (7)$$

yielding:

$$F_x = v_0C_{o,x} + v_1C_{1,x} + v_2C_{2,x} + v_3C_{3,x} \quad (8)$$

The u_i and v_i components can be expressed in terms of the input samples sample $f_0 \ldots f_7$ as:

$$u_i = f_i + f_{7-i} \quad and \quad v_i = f_i - f_{7-i} \; for (i = 0,1,2,3) \quad (9)$$

The 1D-DCT equations can now be written as follows:

$$F_x = \sum_{i=0}^{3} C_{i,x}u_i \qquad (x = 0,\ 2,\ 4,\ 6) \quad (10)$$

$$F_x = \sum_{i=0}^{3} C_{i,x}v_i \qquad (x = 1,\ 3,\ 5,\ 7)$$

In this particular study, the implementation involved the distributed arithmetic method based on Richardson's depiction for 1D-DCT[6]. This is where the modified input samples u_i and v_i are written in a B-bit two complement number. Thus, the previous equation 10 can be expressed in the following form:

$$F_x = \sum_{i=0}^{3} C_{i,x}\left(-u_i^0 + \sum_{j=1}^{B-1} 2^{-j}u_i^j\right) \text{ when x = 0,2, 4, 6} \quad (11)$$

$$F_x = \sum_{i}^{3} C_{i,x}\left(-v_i^0 + \sum_{j}^{B-1} 2^{-j}v_i^j\right) \text{ when x = 1, 3, 5, 7}$$

These F_x equations for x even and odd can be combined to yield:

$$F_x = -\sum_{i=0}^{3} C_{i,x}u_i^0 + \sum_{j=1}^{B-1} 2^{-j} \sum_{i=0}^{3} C_{i,x}u_i^j \quad (12)$$

By defining a new D_x term as:

$$D_x(u^j) = \sum_{i=0}^{3} C_{i,x}u_i^j$$

and by substituting into equation 12, the following equation is derived:

$$F_x = \sum_{j=1}^{B-1} 2^{-j} D_x(u^j) - D_x(u^0) \text{ for x = 0,2, 4, 6} \quad (13)$$

Similar reasoning can be used with regards to each $D_x(v^j)$ components and odd F_x. These equations are used to formulate the concept known as the *distributed arithmetic method*. The coefficients of each $D_x(u^j)$ and $D_x(v^j)$ are pre-calculated and stored in the ROM look-up table for each x value. In our case, we will have 8 ROM Lookup Tables depicting the following arrays:

even rows: $D_0(u_j)$, $D_2(u_j)$, $D_4(u_j)$, $D_6(u_j)$

odd rows: $D_1(v_j)$, $D_3(v_j)$, $D_5(v_j)$, $D_7(v_j)$

For example, $D_0(u_j)$ is written in a 4.11 float-to-fix representation of 15 bits. The upper 4 bits represent the integer component and lower 11 bits represent the fractional component. The upper four bits are the signed integer portion of the number and the lower bits are the fractional portion. After pre-calculations of the ROM lookup table coefficients, SC_MODULE constructor for each individual $D_x(u_j)$ is designed. Each SC_MODULE depicts the individual modules constructor, sensitivity list, and interface input/output (I/O) streams in the header file. The following is an excerpt of how $D_0(u_j)$ is calculated based on equation (12):

u(int)	u(binary)	D_0(float)	D_0(hex)
(0)	0000	0.000000	0x0000
(1)	0001	0.707107	0x05a8
(2)	0010	0.707107	0x05a8
(3)	0011	1.414214	0x0b50
(4)	0100	0.707107	0x05a8
(5)	0101	1.414214	0x05a8
…	….	….	….
(15)	1111	2.828427	0x16a0

Figure 4 depicts the SystemC RTL of the ROM0 header file:

```
#include "systemc.h"
SC_MODULE(rom0){
        sc_in<sc_uint<4> > addr;
        sc_out<sc_uint<15> > data;
        void prc_rom0();
        SC_CTOR(rom0){
                SC_METHOD(prc_rom0);
                sensitive << addr;
        }
};
```

Figure 4: SystemC RTL ROM0 Header File

The member functions of the SC_MODULE are written in a separate source code file. Note that the bit precision of the interface input/output streams can be depicted using SystemC interface sc_in and sc_out connections. In this case, the member function prc_rom0 is a process function sensitive to the change of the address input port as shown in Figure 5. The lookup table is setup for accessing 16 different ROM0 coefficients depicting u_j. The pre-calculated float-to-fix numbers are set to 15 bits of precision. The

interface *addr* is read and output data is retrieved from the ROM0 lookup table.

```
#include "rom0.h"
void rom0::prc_rom0()
{
        sc_uint<15> rom0[16];
        rom0[0]  = (sc_uint<15>)0x0000;
        rom0[1]  = (sc_uint<15>)0x05a8;
        rom0[2]  = (sc_uint<15>)0x05a8;
        rom0[3]  = (sc_uint<15>)0x0b50;
        rom0[4]  = (sc_uint<15>)0x05a8;
        rom0[5]  = (sc_uint<15>)0x0b50;
        rom0[6]  = (sc_uint<15>)0x0b50;
        rom0[7]  = (sc_uint<15>)0x10f8;
        rom0[8]  = (sc_uint<15>)0x05a8;
        rom0[9]  = (sc_uint<15>)0x0b50;
        rom0[10] = (sc_uint<15>)0x0b50;
        rom0[11] = (sc_uint<15>)0x10f8;
        rom0[12] = (sc_uint<15>)0x0b50;
        rom0[13] = (sc_uint<15>)0x10f8;
        rom0[14] = (sc_uint<15>)0x10f8;
        rom0[15] = (sc_uint<15>)0x16a0;
        data.write(rom0[(int)addr.read()]);
}
```

Figure 5: ROM0 RTL Source Code

These ROM modules are connected to the Shift-Accumulator units. Based on the number of B-bits being processed the corresponding coefficient is read from the ROM loop-up table. A shift and add is performed to generate an accumulated stored data. As a result, there will be 8 ROM modules in parallel, shift-accumulator modules, and a bit slice data module (controls B-bits). Each of the internal modules in the top-level DCT (parent module) are considered child modules which are declared as follows:

ROMs

```
u0_rom_ptr = new rom0 ("u0_rom");
v1_rom_ptr = new rom1 ("v1_rom");
…
v7_rom_ptr = new rom7 ("v7_rom");
```

Shift-Accumulator Modules

```
u0shifter_ptr       = new shifter("u0Shifter");
prev_u0data_ptr  = new reg1("prev_u0data");
u0data_reg_ptr    = new reg2("u0data_reg");
final_u0data_ptr  = new buffer("final_u0data");
…
v7shifter_ptr       = new shifter("v7Shifter");
prev_v7data_ptr  = new reg1("prev_v7data");
v7data_reg_ptr    = new reg2("v7data_reg");
final_v7data_ptr  = new buffer("final_v7data");
```

Bit Slice Modules

```
ubit_slice_data_ptr=new bit_slice_data
("ubit_slice_data");
vbit_slice_data_ptr=new bit_slice_data
("vbit_slice_data");
```

The internal and external signals are not depicted in the above example but connected to the interface modules. The hardware architecture for this rom-shift-accumulator module [6] is illustrated in Figure 6.

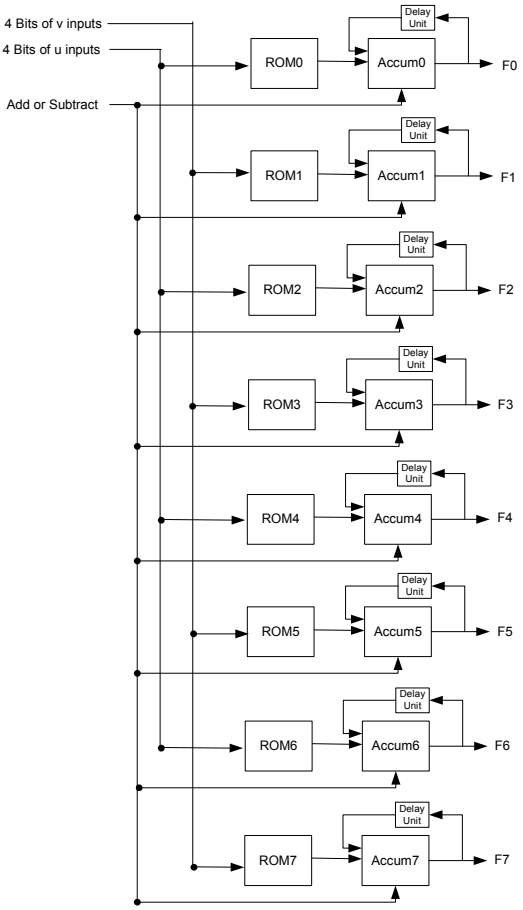

Figure 6.0: ROM-Shift-Accumulator Module

The following is an excerpt of the top-level 1D-DCT Module in SystemC RTL. The header file is shown in Figure 7.

```
SC_MODULE(dct){
    sc_in<bool>  reset;
    sc_in<bool>  clk;
    sc_in<bool>  start;
    sc_in<sc_uint<15> > fx0;
    sc_in<sc_uint<15> > fx1;
    …
    sc_in<sc_uint<15> > fx7;
    sc_out <bool> done;
    sc_out <sc_uint<15> > final_u0data;
```

```
    sc_out <sc_uint<15> > final_v1data;
    …
    sc_out <sc_uint<15> > final_u6data;
    sc_out <sc_uint<15> > final_v7data;
    …
   SC_CTOR(dct){
   …
   }
   ~dct(){
   …
  }
};
```

Figure 7: Top-Level 1D-DCT SystemC RTL

6. Hardware Design: 2D-DCT

Our hardware implementation is based on Richardson's depiction of a hardware implementation of 2D-DCT as shown Figure 8.

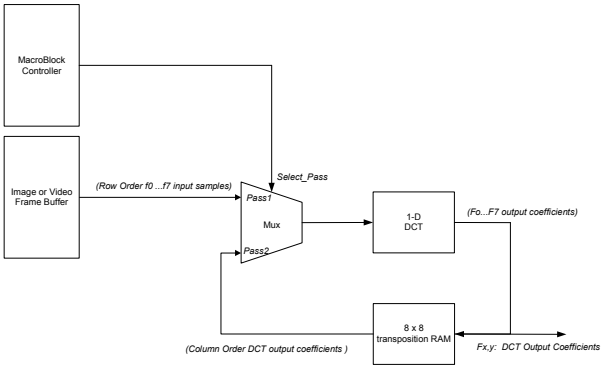

Figure 8: 2D-DCT Hardware Block Diagram

The hardware implementation of 1D-DCT is performed on each row of the 8x8 macroblock being processed. These intermediate coefficients are then stored in the 8x8 tranposition RAM (a two-dimensional array) in the same row order. Then, coefficients read from the RAM in column order are processed using the 1D-DCT architecture. The final DCT coefficients are then read from the Transposition RAM in row order.
The 2-D DCT transform is performed in two passes using two 1-D DCT transform. The first pass will grab from the frame buffer perform the 1-D DCT on each row and store the temporary coefficients in row order into the transposition RAM (8x8 array). The second pass will read the coefficients stored in the RAM in column order and then perform the 1-D DCT on each column. These coefficients will be stored in column order into the RAM and read in row order as the final DCT coefficients outputs. The transposition RAM is composed of an 8x8 array with *n*-bits of data. The determination factor of *n*-bits is specified by the target physical area design constraints of the hardware digital system. The transposition RAM is designed to read and write in column or row order. [6]

7. SystemC Implementation of 2D-DCT Co-Software/Hardware

In our case study, the top level interface signals, internal signals, modules, monitor units, and tracker files are interconnected in the top level SystemC Main Module. The interface signals are composed of all the I/O signals connecting the top level modules. The internal signals are signals internal to the top level modules. These signals can not be access by outside modules. The modules are the representational codes of RTL/TL codes. The RTL code may represent synthesizable code. The monitor files are files which dump information in a specified text format. An the tracer file is a file format used to represent I/O signals in a waveform viewable format. This is represented in Figure 9.

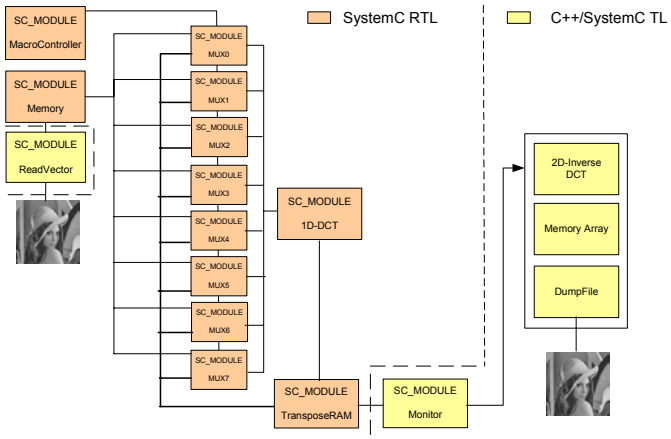

Figure 9: Top-level Functional Verification SystemC Block Diagram

In this case, we dump VCD files tracking interface input/output (I/O) signals representing time events. The VCD files can be viewed using a VCD WaveForm Viewer. In our case, we used SynaptiCAD WaveForm Viewer. The following approach is used to create a SystemC VCD trace file:

```
sc_trace_file *tf
=sc_create_vcd_trace_file("MyDCT_Stimulus");
```

The top-level signals do not need the full path of the signal to be represented such as:

```
sc_trace(tf, start, "start");
sc_trace(tf, clk, "clk");
```

```
int sc_main (int argc, char* argv[]){
//Interface Signals
…
//Internal Signals
…
//Interface Modules
…
//Monitor Modules
…
//VCD Trace Files
…
}
```

Figure 10: SystemC top level sc_main file

The internal module interface can be tracked if references are pointing to the child modules as shown below:

```
sc_trace(tf,dctm.fx0_ptr->reg_out, "fx0_reg");
sc_trace(tf,dctm.fx1_ptr->reg_out, "fx1_reg");
sc_trace(tf,dctm.fx2_ptr->reg_out, "fx2_reg");
sc_trace(tf,dctm.fx3_ptr->reg_out, "fx3_reg");
sc_trace(tf,dctm.fx4_ptr->reg_out, "fx4_reg");
sc_trace(tf,dctm.fx5_ptr->reg_out, "fx5_reg");
sc_trace(tf,dctm.fx6_ptr->reg_out, "fx6_reg");
sc_trace(tf,dctm.fx7_ptr->reg_out, "fx7_reg");
```

Likewise, the VCD tracer file needs to be closed as follows:

```
sc_close_vcd_trace_file(tf);
```

The VCD files will help in the debugging process of the co-design system since we have visual representation of selected interface signals in time-event representation [7][8].

8. Conclusion

This research work addressed a new implementation of the Discrete Cosine Transform (DCT) in a co-hardware/software design environment using SystemC, a new state-of-the-art hardware description language (HDL). New standards and functional hardware verification methodologies are being developed on this next-generation standard. This co-design environment will be used for image and video SystemC architectures. In our case study, our modeling methodology introduces the capability of exploring feasibility studies in a C++/SystemC Transaction level modular designs and refining these components once fully functional into RTL modeling. In the proposed architecture, the 2D-DCT was designed in a RTL modeling style while surrounding modules where designed in a C++/SystemC TL. Monitors files are used to dump text file format representation of vital information for the verification process. Likewise, VCD trace files where used to visual interface signals time events in a waveform viewer. The framework thus exploits the true features that SystemC could bring to a real-time co-design environment in industrial designs. Efforts are being made to expand the modeling framework of the image/video co-design.

References

[1] J. Bhasker, **A SystemC Primer**. Star Galaxy Publishing, Allentown, PA, 2002.

[2] Thorsten, G. et al. **System Design with SystemC**. Kluwer Academic Publishers, Boston, 2002.

[3] Ming-Hau Lee, Design and Implementation of the High-Performance Low-Power MorphoSys Reconfigurable Processor, UMI, Ann Arbor, MI, 240 pages, 2002.

[4] Dan Crisu, A Hardware/Software Co-Simulation Environment for Graphics Accelerator Development in ARM-Based SOCs, The Netherlands, 2002.

[5] Ali Sayinta, Gorkem Canverdi, Marc Pauwels, Amer Alshawa, Wim Dehaene, A Mixed Abstraction Level Simulation Case Study Using SystemC for System on Chip Verification, Germany, 2003

[6] Iain Richardson, Video Codec Design. JohnWiley & Sons, Inc, New York, New York, 299 pages, 2002

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

[8] Synopsys Inc. *SystemC Version 2.0 Users Guide*. 2003 at: www.systemC.org.

Acknowledgements

This research was supported by the National Science Foundation Grants EIA-9906600, CNS 042615, and HRD-0317692; and the Office of Naval Research Grant N00014-99-1-0952.