Transaction-level Modeling of MPEG-2 Video Decode Application in SystemC 2.0.1

Samvit Kaul, Sreekanth M, Junhyung Um, Eui-young Chung,
Kyu-Myung Choi, Jeong-Taek Kong, Soo-Kwan Eo
[†]CAE Center, System LSI Devision, Device Solution Network, Samsung Electronics
Email: {samvit.kaul, sreekanth.m, euiyoung.chung, kmchoi, jkong, sookwan.eo}@samsung.com

Abstract — *We present our transaction level model of the MPEG2 Video Decoder implemented in SystemC 2.0.1. This model can serve as a basis for architecture design-space exploration, hardware/software partitioning, functional validation, and performance evaluation. It can be viewed as a typical example of high level modeling in SystemC, and the results can be applied to most SystemC based models which use blocking threads and dynamic sensitivity as their modeling methodology for rapid system prototyping.*

Simulation Speed results are presented in both un-timed and timed simulation models of MPEG-2 decoder application using various characteristic test streams. Empirical evidence is found for the fact that the speed of a timing simulation of complex systems using SystemC threads and dynamic events depends on the timing parameters of blocking threads.

I. Introduction

NOW platforms are being defined which include a wide assortment of elements from System-level design (SLD): the RTL hardware definition, bus architecture, power management strategy, device drivers, OS ports, and application software. However, to be successful, a platform will need more than this. An essential element for enabling differentiation will prove to be an advanced systems modeling and verification environment. Developers require a variety of views of the entire platform from RTL, system models, software development models, and real hardware development boards. That means, system views must be extendible, allowing designers to exploit the advantages of a well-supported, pre-verified base platform of hardware and software IP, while differentiating their own application with their own IP. Specifically, each design task has specific requirements on methodologies and IP customers will want to make extensions to the IP during each stage of their own design.

In addition, at the system-level, availability of software becomes critical and it is no longer reasonable for the software team to wait for a prototype system. Coverification can move the integration schedule forward to the point where RTL is available, but this still delays the software integration to a point where much of the hardware design is complete. System and software designers would still be lacking a common environment.

SystemC provides a solution to this dilemma. It is the standard design and verification language that spans from concept to implementation in hardware and software and which can also be used to develop models.

One of the key techniques used in this design flow is the modeling of the system at the transaction level. Transaction level modeling (TLM) is simply a higher abstraction level for modeling. Systems modeled in RTL are concerned about the hardware details such as pin-level behavior of their system. With TLM, it is possible to accurately model many aspects of a system at a higher (e.g. Read and Write) level. By using TLM we are simplifying the modeling effort and we also gain simulation speed.

In this paper, we present our implementation of the MPEG2 MP@ML Video decoder using SystemC 2.0.1. Designing such a system, with possible support for multiple standards and future proof scalability, performance requirements on cost and power, is a non-trivial job. Currently the solutions present a staggering range [1]-[5]. The design space is huge, and one is forced to look for point solutions depending on an possible application area. One approach to handle the design complexity is to follow a system-level design methodology, which enables design decisions to be taken very early in the design stage, and realization of their consequences in a rapid prototype.

The outline of the paper is as follows. In Section II, we present a quick synopsis of SystemC, and transaction-level modeling. In Section III, we give a short background on MPEG-2 Video. Section IV presents some details on transaction level modeling and Section V presents our implementation of the transaction level model of MPEG-2 Video Decoder in SystemC. Section VI presents the results and Section VI concludes the paper.

II. System Background

SystemC [6]-[8] is an object oriented C++ library that enables a designer to specify, implement and verify a design that spans from concept to implementation in hardware and software. The Open SystemC Initiative (OSCI) is a consortium of major EDA and IP companies that contributes to and governs SystemC development and distribution. SystemC users may develop models using SystemC along with standard ANSI C++ compilers. In recent times, SystemC has emerged as the de-facto standard for system level design

efforts, replacing hitherto propriety solutions.

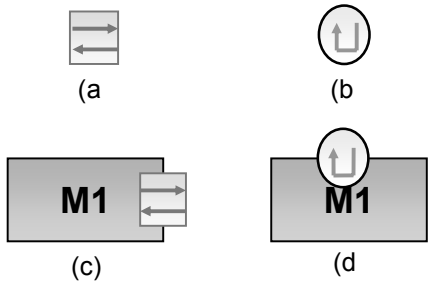

Figure 1 Basic Concepts in SystemC and associated Syntax : (a) A port ; (b) An Interface ; (c) A Module with a port ; (d) A Channel implements an interface MPEG Video

In SystemC a system is modeled as a collection of **modules** connected together in a netlist. The modules may be hierarchical, that is the modules may themselves contain a netlist of modules. A module contains **processes,** which define its behavior and provide a method for expressing concurrency. Also a module contains **ports**, through which the module communicates with the external environment.

Each port has an associated **interface**. The interface defines the valid communication patterns available to a module through its port. A module can only invoke the communication methods that are available in the interface associated with its port.

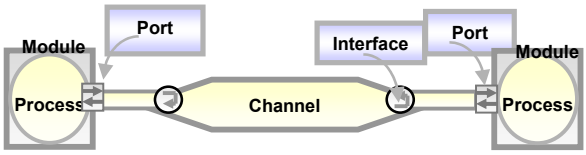

Figure 2 A Simple Netlist in SystemC

A **channel** implements one or more interfaces. A module port is *bound* to a channel, which implements its interface. A netlist of modules is formed by binding all module ports to appropriate channels, thereby making it possible for the modules to communicate to each other.

A toplevel netlist can be executed in simulation by the SystemC Simulation Kernel, which implements an **event-driven simulation** framework. **Events** are first class objects available to the SystemC programmer, which can be used for explicit module *synchronization*. SystemC simulation is not started until all module ports are properly connected and initialized. The visual syntax used to describe these architectural SystemC structures is depicted in Figure 1. A Simple module netlist with the module ports bound to a channel is depicted in Figure 2.

.

III. MPEG VIDEO

The Moving Picture Experts Group (MPEG) standardization is responsible for creating the most popular standards for digital audio-video compression [9]-[11].

MPEG-1, defined audio and video compression coding methods and a multiplexing system for interleaving audio and video data. MPEG-1 principally supports video coding up to about 1.5 Mbit/s giving quality similar to VHS and stereo audio at 192 bit/s.

The MPEG-2 standard is capable of coding standard-definition television at bit rates from about 3-15 Mbit/s and high-definition television at 15-30 Mbit/s. MPEG-2 extends the stereo audio capabilities of MPEG-1 to multi-channel surround sound coding. MPEG-2 decoders also decode MPEG-1 bitstreams. MPEG-2 aims to be a *generic* video coding system supporting a diverse range of applications.

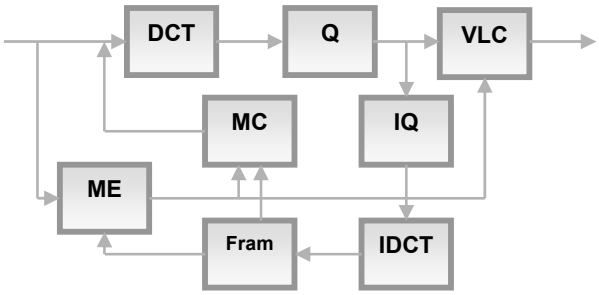

Figure 3 The MPEG Video System

The MPEG standard achieves compression by exploiting the spatial and temporal correlation between the pixels values in a digital video signal. In addition, it also exploits the psycho-visual redundancy as perceived by the human visual system. The human eye has a limited response to fine spatial detail, and is less sensitive to detail near object edges or around shot-changes. Consequently, controlled impairments introduced into the decoded picture by the bit rate reduction process should not be visible to a human observer.

Two key techniques employed in an MPEG codec are intra-frame Discrete Cosine Transform (DCT) coding and motion-compensated inter-frame prediction.

IV. TRANSACTION LEVEL MODELING(TLM)

The fundamental concept in SystemC modeling is the separation of module behavior, and module communication. A designer could independently focus on the behavior of the module, or the communication systems used by the module, and still be able to integrate seamlessly the two together. In the parlance of object-oriented engineering, we say that the module communication is abstracted to an interface rather than to an implementation. SystemC Channels are abstractions for module communication and SystemC Module Processes are abstractions for module behavior.

Transactions are high-level communication abstractions provided to modules through a port-interface. The high level is in the sense that the focus is not on pin-level as is done in typical hardware design, but rather the focus is on protocol objects (address, data, handshake, control signals) and

protocol semantics (blocking, non-blocking, cacheable, arbitrated etc.).

Transactions in SystemC 2.0 are Channel implementations of specific interface functions. The functions typically implement a communication mechanism (peer-to-peer, shared memory, bus interconnect, etc.) and need to be called in a specific way by the module, which intends to use them.

V. TLM OF MPEG-2 VIDEO DECODER

We modeled the MPEG-2 decode application as a *Macroblock-Pipeline*. The modules are independent, possibly concurrent entities, communicating as a dataflow network [12]-[13]. There is no explicit synchronization between modules except wherever necessary by the algorithm (e.g. Header Parsing and Variable Length decoding). This enables us to make all the inherent parallelism of the algorithm explicit in the model, which can be exploited by a system architect in an implementation. Since we are concerned with modeling a system-level application model, we do not force a module behavior (Hardware/Software) or a communication-architecture (e.g. a shared bus) in the model. This allows for orthogonal design choices between module behavior and communication in design space exploration. Following are the details of the model in execution. The inter-module-communication in the macroblock pipeline is implemented as blocking read-write transactions.

BitstreamInput is a hierarchical channel which implements the ***BitstreamInputInterface*** providing the well-known MPEG primitives *getbits(n), showbits(n), flushbuffer(), and next_start_code()*. ***HeaderParser*** module uses *BitstreamInput* to fetch the compressed bitstream and parses it for the header and extension information as described in the standard. As soon as the headers are parsed, *HeaderParser* triggers the ***VLCParser*** module to start the variable length decoding of the current macroblock. All ***InterModuleCommunication*** in the macroblock-pipeline happens through a hierarchical channel which implements many interfaces which define communication between individual modules. For example, the communication between *HeaderParser* and *VLCParser* is called ***HeaderVLCInterface*** and provides mechanism for communication of all the parameters and control information from *HeaderParser* to *VLCParser*. Similarly, there are other interfaces like *VLCIQInterface*, *VLCMCInterface*, IQIDCTInterface etc.

Once the VLCParser finishes decoding the current macroblock, it forwards the relevant data and control information to ***InverseQuantization*** module and ***MotionCompensation*** Module. The *InverseQuantization* in turn forwards data and control to the ***InverseDCT*** module. A macroblock is reconstructed in the ***FrameBuffer*** module, which accepts relevant data and control information from *MotionCompensation* and *InverseQuantization* modules. The *FrameBuffer* module can optionally output the reconstructed *frame* to a target (X-Display, Unix File, a SDRAM model).

Each module has one or more SC_THREAD processes which implement the behavior of the module. Typically, a module process makes a blocking interface call to input data, processes the data, and makes a blocking call for outputting the data. In case there is no serial data-dependency between modules, they are concurrent in execution. This is depicted in the pseudo-code below:

```
//Typical Module Thread Process
while(1) {
   ModulePort->getdata();            //blocking call
   ModulePort->getcontroldata();     //blocking call
   ProcessData();
   if (TimedMode){
      wait(ProcessCycleCount);
   }
   ModulePort->putdata();            //blocking call
   ModulePort->putcontroldata();     //blocking call
}
```

The inter-module-communication transactions are implemented as blocking interface methods inside the hierarchical channel, and uses dynamic sensitive events for achieving blocking effects. Dynamic sensitivity of threads to events, ensures that the blocked threads are not un-necessarily called by the simulation kernel, thereby providing an efficient scheme for simulation.

The threads can have timed or un-timed operation depending on a command line configuration parameter, which controls whether the module blocks for a predetermined

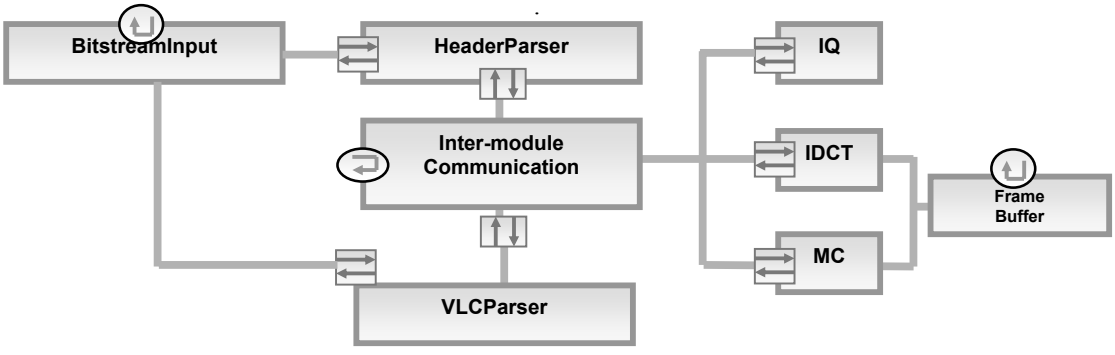

Figure 4 Transaction-level Model of MPEG-2 Video Decoder

number of cycles accounting for module behavior. Currently, we specify the following timing parameters in the timing mode:

- **PerByteHeaderParserTime** specifies the processing time for each byte of a MPEG Video Header. For example, if the sequence header size is 12 bytes, then it would consume (PerByteHeaderParser * 12) Cycles to process the sequence header in simulation. This makes the processing time *data-dependent*.
- **PerBitVLCParserTime** specifies the processing time for each bit of a MPEG Macroblock. For example, if the macroblock size is 150 bits, then it would consume (PerBitVLCParser * 150) Cycles to process the macroblock in simulation. This makes the processing time *data-dependent*.
- **PerBlockIQTime** specifies the processing time for a 8x8 block Inverse Quantization. For a given sequence chroma format, the total processing time for a macroblock is (Block_Count * PerBlockIQ) cycles.
- **PerBlockIDCTTime** specifies the processing time for a 8x8 block Inverse Discrete Transform. For a given sequence chroma format, the total processing time for a macroblock is (Block_Count * PerBlockIDCT) cycles.
- **PerMBlockForwardMCTime** specifies the processing time per macroblock Forward Motion Vector Prediction.
- **PerMBlockBackwardMCTime** specifies the processing time per macroblock Backward Motion Vector Prediction.
- **PerMBlockReconTime** specifies the processing time per macroblock reconstruction, saturation, and post-processing.

Specifying different timing parameters specifies different points in a design–space, which can be traversed and explored for a particular cost-performance tradeoff. Figure 4 captures the essence of these details.

VI. Experimental Results

The model as described was implemented using the OSCI SystemC 2.0.1 library, compiled with GCC 2.95.2 at –O3 compiler optimization level and simulated on a Sun-Blade 1000 machine. The model borrows a lot of code from an open-source public-domain, C based MPEG-2 Video Decoder, *mpegdecode [14]*. Substantial portion of the code was refactored and freshly implemented. Most of the refactoring comes from removing the global data structures of *mpegdecode*, and creating the SystemC modules and channels.

Two sets of timing parameters were used as example implementations in HW/SW combinations. Here are the two sets:

Set 1

Timing Parameter	Value (Cycles)
PerByteHeaderParserTime	1.0
PerBitVLCParseTime	0.25
PerBlockIQTime	10
PerBlockIDCTTime	10
PerMBlockForwardMCTime	10
PerMBlockForwardMCTime	10
PerMBlockReconTime	2

Set 2

Timing Parameter	Value (Cycles)
PerByteHeaderParserTime	1.0
PerBitVLCParseTime	0.125
PerBlockIQTime	100
PerBlockIDCTTime	10
PerMBlockForwardMCTime	100
PerMBlockForwardMCTime	100
PerMBlockReconTime	50

Four MPEG-2 streams were used to test the simulation speed. All four streams were in the chroma format 4:2:0. The details of the streams are tabulated.

Video Sequence	Total Number Of Frames	Frame Size	Picture Sequence
Trees	61	720 x 480	Interlaced
Hop	31	720 x 576	Progressive
Ballerina	60	352 x 224	Mixed
Conference	20	720 x 576	Progressive

Speed measurements were made using the UNIX *time* command. In the following table the *User Time* is reported in seconds. The three simulation options of the model are untimed mode (UT), timed mode with timing parameters as set-1 (TM-1), and timed mode with timing parameters as set-2 (TM-2).

	Trees	Hop	Ballerina	Conference
UT	40	24	9	15
TM-1	79	47	16	26
TM-2	184	106	39	65

This empirical data suggests that for sequences of comparable dimensions (Trees, Hop, Conference), the un-timed simulation runs at approximately 1.3 frames per second, whereas for the sequence with approx. five times smaller dimensions (Ballerina), the simulation runs at 6.7 frames per second, i.e. 5x faster. Therefore, we can expect an un-timed

simulation of a HDTV (1920 x 1152, 40 frames per second) application to run at approximately 0.24 frames per second, which translates to a throughput of approximately 170 seconds for simulating 1 second of video.

On the other hand, the simulation performance of timed model is dependent on the timing parameters. For the data of set-1, we see that the throughput for 1 second of HDTV video is approximately 350 seconds. Similarly, for set-2, the number comes to approximately 750 seconds. Assuming that set-1 and set-2 represents the end points of a design space to be explored, we can interpolate to estimate our simulation effort for performance evaluation of design options.

VII. Conclusion

We conclude that SystemC presents a convenient method of modeling complex applications for design space evaluation. The simulation kernel gives good results for un-timed simulation model. However, for timed models, one needs to be careful because the simulation speed depends on the timing parameters, which control the triggering of the threads in the system. If there is large variation in the timing parameters, the simulation speed is appreciably affected.

References

[1] Hiroshi Okano, et. al, "An 8-Way VLIW Embedded Multimedia Processor Built in 7-layer Metal 0.11um CMOS Technology", IEEE International Solid State Circuits Conference, 2002.

[2] M. Sima et. al, "*MPEG Macroblock Parsing and Pel Reconstruction on an FPGA-augmented Trimedia Processor*", IEEE International Conference on Computer Design, Austin, Texas, 2001.

[3] M. Ikeda et. al, "*An MPEG-2 Video Encoder LSI with Scalability for HDTV based on Three-Layer Cooperative Architecture*", Design Automation and Test in Europe Conference, 1999.

[4] Edgar Holmann, Toyohiko Yoshidam Akira Yamada, Shin-Ichi Uramoto, "*Single Chip Dual Issue RISC Processor for Real Time MPEG-2 Software Decoding*," Journal of VLSI Signal Processing, 18, 1-13, 1998.

[5] Masaki Toyokura, Hisahi Kodama, et. al, "*A Video DSP with a Macroblock Level Pipeline and a SIMD Type Vector Pipeline Architecture for MPEG2 CODEC*," *IEEE Journal of Solid State Circuits.*, Vol. 29, No. 12, December 1994.

[6] Open SystemC Initiative (OSCI) Website: http://www.systemc.org.

[7] Thorsten Groetker, Stan Liao, Grant Martin, Stuart Swan, "System Design with SystemC", Kluwer, 2002.

[8] Functional Specification for SystemC 2.0, January 2001. Available from OSCI at http://www.systemc.org.

[9] International Standard ISO/IEC IS-13818*: Generic Coding of Motion Pictures and Associated Audio*, Part 2: Video.

[10] International Standard ISO/IEC IS-11172: *Generic Coding of Motion Pictures and Associated Audio*, Part 2: Video.

[11] P.N. Tudor, "*MPEG-2 Video Compression*", Electronics & Communication Engineering Journal, December 1995.

[12] E.A.de.Kock G. Essink, et. al, "*YAPI: Application Modeling for Signal Processing Systems*," *Proceedings of the Design Automation Conference*, June 2000

[13] Pieter van der Wolf, et. al, "*An MPEG-2 decoder case study as a driver for a system level design methodology*", Proceedings of the seventh international workshop on Hardware/software co-design, Rome 1997**.**

[14] MPEG Software Simulation Group, "*MPEG-2 Video Decoder*", Version 1.1, June 1994