SoC Design Case Study Using SystemC Specifications

F. Abbes, E. Casseau, M. Abid

Abstract**-- Modern systems become more and more complex and tendency turn to the integration on one single chip: System on Chip (SoC). A major constraint consists of "Time-to-Market". Hence, the emergence of SoC is creating many new challenges, especially, the necessity of a unified language for the system level design. SystemC is proposed as a standardized modeling language intended to enable system level design at multiple abstraction levels for hardware / software systems. This paper describes a method of stepwise refinement with SystemC, starting from an algorithmic description and progressively adding implementation details. The method is described with reference to a Turbo encoder, which is progressively moved from a purely abstract level to a more detailed description. This study is realized to emphasize on the importance of this tendency within the framework of SoC design. We also present the experimental results from specification, refinement and validation with SystemC and simulation effectiveness of the proposed method.**

Index Terms**-- SoC, System Level Modeling, SystemC, Refinement Methodology, Turbo Code.**

I. Introduction

Design exploration for SoC designs requires a high-level language to provide a model of the whole system. This model should express the design functionality at a sufficiently abstract level so the design is easy to understand and verify [1].

Furthermore, if the SoC design models can be simulated rapidly, then the whole design process can be accelerated. The current approach to system level design uses a mixture of design languages. Each language is used for the aspect of the system to which it is best suited: C/C++ may be used for the system level analysis and software development process. VHDL [2] or Verilog [3] can be used for hardware design.

This approach is time consuming so there is a need for a unified modeling language to rapidly design a whole system.

In response to these needs, the Open SystemC Initiative (OSCI) [4] was announced in September 1999. It is a modeling platform consisting of C++ class libraries and a

simulation kernel [5]. SystemC is a language based on C++, meant to represent functionality, communication, software and hardware at various system levels of abstraction.

Besides, a market for all these researches is that of telecommunications which represents one of the most interesting applications of co-design because it implies the production at a lower cost of systems with strict constraints and this must be made in an environment of big competition where a short time of launch on market is crucial [6].

In this paper, we briefly describe existing modeling languages approaches for the design of a system at a high level of abstraction, in particular the SystemC Initiative. The

work is done as a case study with focus on the SystemC modeling and simulation effectiveness for telecommunication systems. In order to estimate the associate flow methodology, we considered the Turbo encoder technique according to different considered level of abstraction iterating appropriate elementary refinements. We describe the different refinement steps with SystemC, starting from a purely algorithmic description and progressively adding details about data flow, data representation, algorithm timing and scheduling.

The objective of this work is to evaluate the simulation performance of SystemC 2.0 using different ways of modeling at different levels when the complexity of the model increases. We restricted our development experimentation to the hardware flow.

This paper is organized as follows. Section 2 analyzes the need of system level specification in the co-design flow and summarize various existing approaches for the implementation of a system modeling language, especially the SystemC approach, on the one hand, and outlines elementary refinements to consider at different levels of abstraction on the other hand. Section 3 describes the Turbo encoder function at various conception steps. Section 4 gives the experimental results from specification, refinement and validation with SystemC and simulation performance. Finally, section 5 concludes this paper.

II. System Level Design Requirements

With the increasing complexity of today's systems and the move towards SoC, there are growing needs for system-level modeling languages that can be used to describe systems at a high level of abstraction. In order to improve time-to-market and help to simplify the design process, it would be helpful if a single high-level model of the system could be used to implement the whole system.

Manuscript received August 14, 2003.

F. Abbes is with the GMS, ENIS, 3038 Sfax, Tunisia (e-mail: abbes@iuplo.univ-ubs.fr).

E. Casseau is with L.E.S.T.E.R., BREST, Road Saint Maude - 56100 Lorient, France (e-mail: emmanuel.casseau@univ-ubs.fr).

M. Abid is with the GMS, ENIS, 3038 Sfax, Tunisia (e-mail: mohamed.abid@enis.rnu.tn).

A. *The Need For System Level Design*

System level design issues are becoming increasingly critical as implementation technology involves more and more complex integrated circuits and software programs. Mixed-language system level design flows don't allow rapid exploration of the design space or a unified specification and modeling language. There is an obvious advantage in the use of a single language for the system level model, which can also describe Intellectual Property (IP) blocks, system hardware and software. So that, we can easily move functionality between these domains to obtain the best partition from numerous alternatives so any design problems can be resolved much earlier in the design process.

Because there was no existing "perfect" system level design language, three predominant approaches are currently batted around in the marketplace. Indeed, some people suggested developing of totally new languages (such as Rosetta and Superlog) [7]. Other people suggested reshaping the C++ programming language to enable it to describe hardware concepts (SystemC for example). The third approach extends the existing HDLs upwards in abstraction, so they can express system-levels concepts. Each approach has its strengths and weaknesses at various points along the path from high level design description down to RTL [8].

Since SoCs are often 80% software and 20% hardware on average and C/C++ has traditionally been used for programming, the C/C++ derivatives are considered the best approach for full system-level work [7]. However, a system approach does not rest only on the language. It is also essential to have a support in terms of methodology and an availability of EDA tools [9]. Hence, OSCI seems to be more favored with regard to the other approaches as far as the main EDA companies support it. To examine the feasibility of this approach, we need to examine the features and characteristics of SystemC.

B. *SystemC Approach*

SystemC introduces many concepts to support the material modeling and description and its inherent characteristics such as concurrency, temporal aspect and features for generalized modeling of communication and synchronization.

To begin, let us introduce some basic SystemC concepts and nomenclature. A system may be modeled as a set of modules: a basic entity in SystemC that contain processes, ports, channels, and even other modules. Channels, interfaces and events are features for generalized modeling of communication and synchronization. Processes define the behavior of a particular module and provide express concurrency. Conceptually, processes execute in parallel, and they may be triggered by various events in the system. SystemC supports three different types of processes- methods, threads and clocked threads. A channel implements one or more interfaces, where an interface is implying a collection of method (a.k.a. function) definitions to be implemented within a channel. A process accesses a channel's interface via a port on the module. SystemC includes a simulation kernel. The kernel contains a scheduler that is responsible for scheduling processes and updating data communication.

The "refinement" is the central concept of the SystemC design flow [5]. Refinement transforms a model to a lower level of abstraction whilst preserving its functionality. A test bench developed, initially at a high level of abstraction may be reused to verify that the design is correctly transformed at each level. The SystemC design flow shows the SystemC models used at each level of abstraction in the refinement approach (figure 1). The highest level is called the Untimed Functional level (UTF). The UTF model describes the system as a data-driven network of processes, which execute in zero time. The next level is characterized by allocating timing to processes (refinement step 1). The explicit modeling defines the timed functional level (TF). Hence, the system model can be partitioned. This involves allocating processes to either hardware blocks or as tasks which run on a system processor.

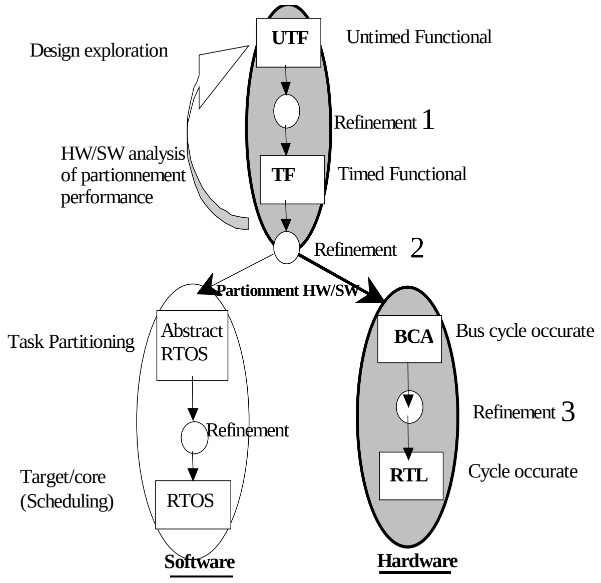

Fig. 1. SystemC Design Flow

Once partitioning has been performed, the hardware component is refined to the Bus Cycle Accurate (BCA) level. This means that the interfaces between the processes may be modeled using a clocked bus cycle model whereas the processes themselves may still execute with zero time delay. Refinement step n° 2 takes the hardware model from a timed functional level to a bus cycle accurate level. The cycle accurate level (CA) describes the hardware in terms of behavior with reference to the system clock.

Step 3 in figure 1 shows refinement from the BCA model to the CA model. The CA level consists of a combination of RTL and implicit state machine modeling. CA models can be used as input to SystemC synthesis tools [10].

However, we should not expect to have a totally automated synthesis tool to generate systematically the design from the abstract specification. In this context, our work consists in focusing in a manual refinement methodology at the systems levels with SystemC. From the system to the implementation level descriptions, the various abstraction levels must be well defined and the refinement steps very precise and clear.

C. *Refinement Using SystemC*

We need a reliable methodology that allows the designer to continue refining the C/C++ executable specification created by the system architect or system level designer into one that is less abstract, i.e. that contains greater details about the structure and operation of the hardware/software it represents. This is usually performed with small steps to keep the process manageable. Each step produces a new model which must be verified to ensure compliance with the original specification and that no design errors have been introduced by the reuse of the original C/C++ testbench. The refinement steps that address these respective aspects are termed as “atomicity refinement”, “algorithmic refinement”, “communication refinement”, and “data refinement”.

The following subsections describe how these refinement steps may be applied in more detail.

1) *Atomicity Refinement (AtR)*

For a SystemC design, the starting point executes a sequential program. Atomicity refinement converts this algorithm into one that contains concurrently executing processes.

2) *Algorithmic Refinement (AlR)*

This is a series of steps for splitting complex tasks into a sequence of smaller so simpler tasks. Algorithmic refinement replaces the functions used in abstracted level programs with a collection of simple functions that can be directly implemented in hardware.

3) *Communication Refinement (CR)*

Communication refinement is a process to replace a primitive channel with a refined channel. A refined channel will often have a more complex interface than the primitive channel previously used.

4) *Data Refinement (DR)*

This is the process to replace abstract data types such as C++ defined type by data structures.

The order and the number of repetition of each of these stages are not congealed; it depends, generally, on the complexity and on the nature of the system to deal with. This could be the model structure, its functionality, communication or internal data representation. It should be noticed that the original testbench is maintained throughout the refinement process. In this way the model functionality is always measured against the original specification. This refinement approach is presented in the following section with the turbo encoder modeling.

III. Experimentation

Error correcting codes are a means of including redundancy in a stream of information bits to allow the detection and correction of symbol errors during transmission. Turbo coding is going to be an important forward-error correcting technique in many of the newer wireline and wireless data-communication systems. There are very powerful forms that bring the performance of practical coding even closer to Shannon’s theoretical specifications [11].

In order to estimate the SystemC methodology, we considered a Turbo encoder as a case study, in particular the PCCC family (Parallel Concatenated Convolutional Code). PCCCs employ two or more recursive systematic convolutional (RSC) encoders joined in parallel by one or more pseudo-random interleavers [12].

A. *Turbo encoder design*

At its conception, this Turbo encoder comprised of the parallel concatenation of two recursive systematic convolutional RSC codes as shown in figure 2 below. An interleaver is used to scramble the order of the input bits before feeding them into the second encoder.

The data bits “input data” are fed into the first encoder which generates a set of systematic and parity bits. The data bits are passed to the second encoder after being permuted by a pseudo-random interleaver. The second encoder also generates a set of systematic and parity bits. Because sending two sets of systematic bits is redundant, the overall code is punctured by deleting the second set of systematic bits.

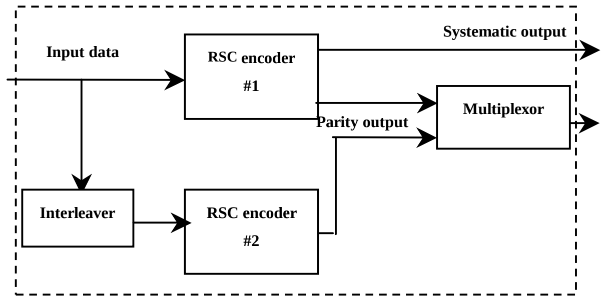

Fig. 2. Turbo encoder Structure

The resulting bit stream consists of a systematic bit from the first encoder followed by the parity bits from the first and second encoders, respectively. Multiplexing the systematic information with the parity information from both RSC encoders produces the output stream of a Turbo encoder.

We assured C++ specification of the application independently from the application properties in term of time, synchronization etc... at first time. It allowed us to ensure functional check by guarding the same sequences to encode.

Details about which refinements steps took place in the Turbo encoder design flow are analysed in the below subsections.

B. *UTF Turbo encoder Model*

At the initial stage of the refinement process, a Turbo encoder C++ algorithm is incorporated into an UTF SystemC model – it is simply placed inside a SystemC module wrapper. This model is used as the reference model for lower abstraction levels in order to evaluate different design properties. Inside a module, the functionality is placed within SystemC Turbo encoder process. The inputs used to drive the algorithm are supplied from a stimulus module (Sc_module *stim*) and the algorithm results are passed to a result module

(Sc_module *res*).

The simulation is event-driven. The event checking mechanism is that of the *sc_fifo* primitive channel available in SystemC 2.0. This channel support data communication between modules. The Sc_module *stim* contains an Sc_thread process that can be suspended and reactivated by events or by module signals. The *res* module has an Sc_method process made sensitive to its input port.

C. *TF Turbo encoder Model*

We proceed to refine the UTF model to the TF model. Many features of SystemC and the C++ cannot yet be implemented in hardware [13]. Therefore, in order to use later synthesis tools, hardware designs must use a restricted subset of SystemC and C/C++.

1) *TFV1(Version 1)*

In the C++ algorithm, a matrix defined data type is used to simplify data manipulation. We affect the named DR to break down this defined data type and so to omit it from the code so that the design hasn't global neither variable nor pointer for dynamic memory allocation. The refined system has been revalidated with the same input sequences to the C++ algorithm.

We add a level of hierarchy to the Turbo encoder module which instantiates two RSC modules, an interleaver module and one multiplexor module. This was achieved by placing each appropriate function for these modules inside a Sc_thread process. Each of the modules was given a timed behavior.

2) *TFV2(version 2)*

There is no particular interest to use a hierarchical organization for this example. This is used when there is an inheritance mechanism which reads the module container to sub-modules. That is why, in this second TF version, the independence of every module is assured and separate module were created as atomicity refinement suggests.

Event mechanism adjusts the processing modules and code was written to interface between the sub-modules. Communication between Turbo encoder system and the stimulus modules (stim, res) modules still as initially modeled using an sc_fifo primitive channel.

3) *TFV3(version 3)*

We consider here the RSC module for refining. At this point in the refinement process, an analysis was performed for a suitable choice of a static representation in the RSC module, so that it receives only one bit as input data. The TF model still retains the data-driven execution for the other modules in the system. Another layer of hierarchy was added after an *AtR*: a RSC module instantiates three XOR port module and two flip-flop port module which include Sc_module processes. RSC is now at a CA model and its data transformations are scheduled with respect to a clock as shown in figure 3.

Data processing is triggered on the positive clock edge when start is asserted. When the process finishes, the done signal is asserted. The RSC module is expressed in clock cycles.

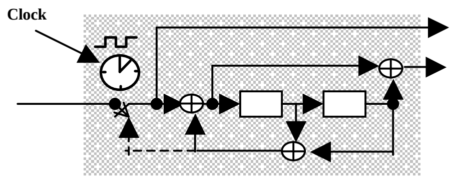

Fig. 3. Refined RSC encoder (dotted lines apply for trellis termination only for decoding)

To co-simulate all Turbo encoder modules, we need to coordinate activation and suspension of *do_interleaver*, *do_stim* and *do_res* processes via operations of data type conversion in appropriate ports and boolean handshaking signals to synchronize bit transferring and data_driven transferring.

IV. RESULTS

Our purpose was to estimate the capacity of SystemC to model, to refine and to validate the conception of an SoC telecommunication function at various levels of abstraction. The modeling and simulation possibilities of this language have been considered with the Turbo encoder case study. This encoder is characterized by the generator matrix of the RSC encoder:

$$G = [1\ 1\ 1,\ 1\ 0\ 1] = [g0,\ g1]=[7,5]_{octal}$$

We check the well RSC functionality by means of GTKwave viewer.

The transfer functions of the 6-state constituent code for PCCC is:

$$G = [1;\ g1\ /\ g0]$$

The initial values of the shift registers of the 6-state constituent encoders are all zeros when starting to encode the input bits.

In order to evaluate the simulation performance of SystemC, for each refinement step previously described, simulations have been done and the associated times have been noticed. These simulation times were obtained with the "time" utility and according to the average afterward 10 simulations for the encoding of the same test sequence of 200 bits. We observe a great evolution of the simulation times with regard to the abstraction levels as shown in figure 4. Indeed, SystemC is based on a real-time kernel giving the designer the possibility to write process definitions with wait-statements and sensitivity lists. Therefore, the SystemC simulation performance is very dependent on the modeling style [14] as well as the full computational model of interaction between the user defined processes and the simulation kernel process. Derived from hierarchically organized modules, SystemC establishes a hierarchical network of a finite number of parallel communicating processes which, under the supervision of the simulation kernel process, concurrently update new values for given signals and variables. Signals do not change their values immediately. Their assignments become effective only in the next simulation cycle [14].

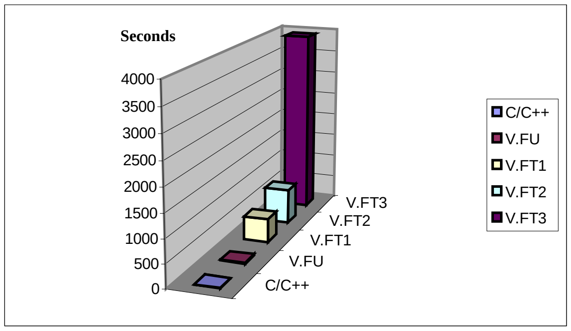

Fig. 4. Turbo encoder simulation Times (Pentium 3, 800MHz 128Mo, Linux Mandrake 8.2)

We notice that versions at high level of abstraction (at the left of the top graphic) take a short simulation time with regard to that at lower levels.

The majority of the simulation time is spent in SystemC's simulator to administer the processes activation as well as to identify eligible processes. When details are added to the SystemC model, then more simulation overhead is incurred. In particular, atomicity refinement slow down simulations because of the increase in the number of SystemC processes. That explains the huge difference in the simulation time between *FTV2* and *FTV3*. The SystemC simulation kernel has to handle this additional workload in addition to an increased number of events. Furthermore, simulation kernel process communicates with the other threads according to internal functions in the used operating system: the realized simulation under an UNIX environment, Linux either Windows NT requires a different time lapse according to the environment characteristics.

If we had still continued to refine the conception, we would have models which should be very long to simulate. With the available tools, it is difficult with so complex applications (Turbo-code for example) to quantify the exact time employed by the scheduler for administering everything.

V. Conclusion

The use of system design language such as SystemC aims at providing a single modeling language for all design abstraction levels. This paper presents our experiments about manual refinements with SystemC. We proceed to the refinement of a convolutional Turbo encoder from a purely functional specification described in SystemC, according to the hardware SystemC design flow. For different refinement steps, SystemC simulation performance has been extracted in order to quantify the simulation time overhead associated to a more refined specification. Although more automation is needed, this work shows that a reasonably efficient implementation can be obtained, allowing for faster system development and quicker time-to-market.

Using this methodology, a Turbo encoder has been created and can be used as an input source file to current SystemC synthesis tools. Further work will focus on the Turbo encoder synthesis to achieve a complete SystemC encoder design.

References

[1] Santarini, Michael. “Million-gate ASICs will require hierarchical flow”, EE Times, http://www.eedesign.com/story/OEG20000120S0052
[2] Alan Fitch, "Application of SystemC to hw/sw co-design". IEEE Seminar - Matériel-logiciel co-design. December 2000.
[3] Matériel Description Languages Compared: Verilog and SystemC, Gianfranco Bonanome, Columbia University, Department of Computer Science, New York, NY.
[4] http://www.systemc.org
[5] Synopsys, SystemC version 2.0 User’s guide.2001
[6] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, pp. 591-600, May 1996.
[7] Pete Hardee " Getting Matériel and Logiciel to Speak the Same Language". Dedicated Systems Magazine pp 6-9, July 2001.
[8] “Design Languages Vie For System-Level Dominance”, Electronic Design Automation 53-60, 1 October 2001.
[9] K.WAKABAYASHI, T.OKAMOTO, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective," IEEE transactions on computer aided design of integrated circuits and systems, VOL.19, NO.12, December 2000, pp.1507-1522.
[10] Economakos G, Oikonomakos P, Panagopoulos I, Poulakis I, Papakonstantinou G, "Behavioural Synthesis with SystemC", Proceedings of DATE-2001, pp p.21-5, 2001.
[11] C.Berrou, A. Glavieux et P. Thitimajshima "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", IEEE International Conference on Communications, ICC-93, May 1993, vol. 2, pp. 1064-1070.
[12] Eric K.hall and Stephan G.Wilson "Stream_Oriented Turbo Codes", IEEE Vehicular Technology Conf. VTC '98 Ottawa, CA, May 1998.
[13] Synopsys, Inc, CoCentric SystemC Compiler, "Describing Synthetisable RTL in SystemC" , January 2002.
[14] Wolfgang Mueller, "The Simulation Semantics of SystemC", DATE 01, Munich, Germany, March 2001.