System-Level Modeling and Design Using SysML and SystemC

Waseem Raslan
Mentor Graphics Egypt
51st Beirut St., Heliopolis
Cairo, Egypt
Email:waseem_raslan@mentor.com

Ahmed Sameh
The American University in Cairo
Computer Science Department
Down Town, Cairo, Egypt
Email:sameh@aucegypt.edu

***Abstract-* SysML, the dedicated system-level UML-based notations proposed by the OMG, is gaining a lot of momentum to be widely accepted by system-level designers. IEEE has standardized SystemC to gain the fame of being powerful at the system-level architectural design and verification. Building on both SystemC and SysML success, we have dedicated this research to study and prototype the automatic translation of SysML designs into SystemC models. This work is an attempt to accelerate the SoC design process by raising the abstraction level in an automated environment.**

I. SYSTEM-LEVEL LANGUAGES

SystemC is an IEEE standard library built on top of the C++ language to help in raising the abstraction level of the hardware description modeling language. The main reason behind building this library on top of C++ language is to leverage on the stability of the C++ development environment tools.

SystemC provided a powerful new hardware abstraction level that is the Transaction Level Modeling or TLM. SystemC provides the following to the designer:

1. SystemC provides a kernel for event-driven simulation that is the base infrastructure to handle events and processes. The main building blocks here are sc_event, sc_process, and sc_thread.
2. SystemC provides predefined hardware specific data types in addition to the ability to add any data type using the C++ ability of inheritance.
3. SystemC provides the concept of channel interfaces and a set of primitive channels that are useful in describing primitive communication between different modules. The main building blocks here are sc_signal, sc_fifo, sc_mutex, and sc_semaphore.
4. SystemC provides a set of very useful libraries to do verification and perform transaction level modeling.
5. Several system level design building blocks. The main building blocks here are sc_module, sc_port, sc_interface, and sc_channel.

On another track, advocates of the Verilog language built another system-level language on top of Verilog and called it System Verilog. System Verilog tried to get all the benefits it can from the different existing languages like; C++, VHDL, Perl, Vera, and PSL. Researchers and designers were arguing on the best uses of System Verilog and SystemC. Arnout and Brophy have proposed in [1] that both languages can live together in a unified design flow. Arnout and Brophy suggested that SystemC best fits for architectural design and verification in addition to HW/SW co-verification. On the other hand they suggested that System Verilog best fits in RTL-to-gates design and verification. Thus, SystemC represents the logical choice to go into lower level of abstraction after the UML/SysML level.

On the other hand, UML proved to be very successful as a software modeling notations in the recent decade. Researchers spent a good deal of HW/SW research efforts to profit from UML success into hardware modeling and design. Nearly each research work has created its own UML profile, through extending the UML, to be able to use the UML notations in describing hardware. The International Council on System Engineering has made a strategic decision in 2001 and collaborated with OMG to standardize a UML profile that fits system-modeling requirements. The result of this collaboration is a UML 2.0 based system-level notations called SysML. UML unified the modeling languages used in the software industry. Similarly, SysML pioneers designed it to unify the diverse modeling languages currently used by systems engineers.

Automatic generation of SystemC code from the SysML specifications will benefit the system design life cycle in the following manner:

1. Filling the abstraction gap between specifications and RTL design cycle as in [11].
2. Increasing the design productivity as in [7]
3. Creating an executable specification that is the golden model of the system design
4. Narrowing the gap between software and hardware designers, in addition to reducing the HW/SW integration and interfacing problems

II. PREVIOUS WORK

Aspiring to create a complete design flow from UML down to the synthesis tools, Rosenstiel et al. have in [12] the Object Oriented Analysis System or OOAS that starts with the UML specifications, generates object oriented C++ code out of this UML specification. Since the synthesis tools are not able to synthesize that C++ code, the OOAS tool analyzes the generated code to generate an abstract syntax tree of the design. Then, OOAS does a class hierarchy analysis of the design followed by an object control data flow analysis. From this information, OOAS creates the control data flow graph of the design, from this graph, OOAS generates C procedural code that should behave like the original object-oriented code. OOAS feeds this procedural code to the down stream synthesis tools that can handle this procedural code.

Mellor Blacer have proposed in [7] an executable and translatable system that allows the designers to model debug, and validate their UML model through model execution regardless of the actual implementation. Mellor et al. are

1-4244-0797-4/07/$20.00 © 2007 IEEE

convinced, as in [8], that only the class, state chart and the action diagrams are fair enough for modeling hardware.

Other research works have been conducted targeting SystemC code generation focusing mainly on the class and state machine diagrams. Butler and Snook have proposed in [3] a formal methodology form UML translation to SystemC through the B formal language. Nguyen et al. have proposed a system in [9] a template-based code generation system that parses the exported XML file that represents the UML design and passes the information to the Velocity engine to generate SystemC code. Riccobene et al. have proposed a UML 2.0 based profile that can easily map to SystemC, as in [11]. Tan et al. have proposed a system called RT2SystemC in [14] that generates C++ code from the UML specifications then the system parses the generated code using JDOM to create SystemC code.

III. CODE GENERATION APPROACHES

To create the translator prototype, we have conducted a survey on the commercial and public share UML and SysML editing tools to select Rhapsody 6.1 tool of I-Logix to build our mapping prototype.

We have investigated many design alternatives to be able to implement the automatic SystemC code generation from SysML designs:

A. *Classic UML model approach*

This approach depends on generating XMI model equivalent representative of the SysML design. The SysML design, now represented in XMI format, can be fed to one of the template-based code generation environments as in [4]. This environment applies appropriate templates on the XMI design to generate user customized SystemC code. We have identified the Eclipse-fostered-project OpenArchitectureWare as a candidate environment to perform the code generation step, but unfortunately, OAW was not ready to communicate easily with Rhapsody 6.1 tool. In addition, Rhapsody supports exporting XMI of UML 1.2 notations. As a result, Rhapsody exports an XMI document that has UML 2.0 and SysML constructs represented in UML 1.2 stereotypes. We expected many problems due to this limitation if we go ahead with the approach.

B. *Eclipse-UML2 model approach*

The Eclipse-fostered UML2 project has created a lightweight complete UML2.0-based data-model that is expected to be adopted as the common data-model of most UML vendor tools. OpenArchitectureWare can read Eclipse-UML2 data-model. OAW can also generate code out of this lightweight data-model via templates as well.

However, unfortunately, Rhapsody 6.1 is not based on the Eclipse-UML2 data-model. Therefore, this approach depends on traversing the Rhapsody data-model through its API's and generate the equivalent Eclipse-UML2 data-model that could be then fed to OpenArchitectureWare. OAW can then apply its SystemC templates for code generation. This approach would have doubled the effort towards generating the SystemC code.

C. *Rhapsody API and a visitor based approach*

This approach depends on accessing Rhapsody 6.1 data-model information through its API interface. Rhapsody provides various ways to access its APIs: through Visual Basic, VB scripts, VC++, and Java. We have selected VC++ to access the API's.

Having accessed Rhapsody SysML design information through its API's, we have selected the visitor-based mechanism, as in [4], to traverse the SysML design data-model and generate the appropriate SystemC model for each component.

Although the third approach is heavily dependant on Rhapsody data-model and API's, we have selected it to conduct this research and produce the SystemC models. Yet, we have designed the translation system in a way that separates Rhapsody data-model traversal from the code generation to allow for immediate Eclipse-UML2 data-model generation in near future. We have created a prototype to show the visibility of the third approach and the suggested mapping. The prototype is able to generate SystemC code for the structure SysML specifications and behavioral SysML specifications. In addition, the prototype uses the redundant information presented in the use case diagrams to validate the integrity of the whole design.

Figure 1 shows an overall idea of the translator relationship with the editing tool while using the third approach.

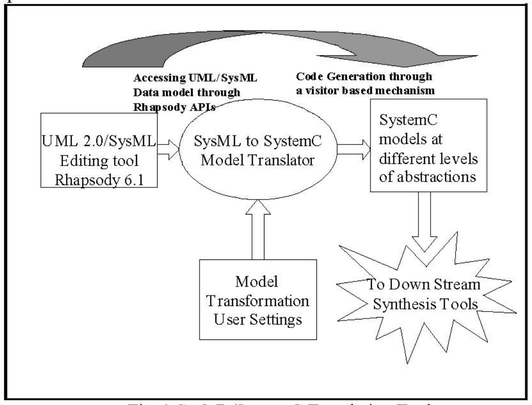
Accessing UML/SysML Data model through Rhapsody APIs
Code Generation through a visitor based mechanism
UML 2.0/SysML Editing tool Rhapsody 6.1
SysML to SystemC Model Translator
SystemC models at different levels of abstractions
Model Transformation User Settings
To Down Stream Synthesis Tools

Fig. 1 SysML/SystemC Translation Tool

IV. SYSML TO SYSTEMC TRANSLATION RESULTS

Through this research work, we have identified how SysML constructs map to SystemC. In addition, we have used the redundant information presented in the sequence diagrams to validate the design integrity. Aspiring to benefit from the SysML different diagrams in visualizing, designing, documenting and validating the SysML model, we have placed no restrictions on the types of the diagrams that will be translated into SystemC. Nevertheless, we have put some SysML design guidelines in place to reduce the design ambiguity that can lead to incorrect SystemC code generation. We have published the mapping details and sequence diagrams are used to validate the design in [10]. We have used the implemented SysML to SystemC translator in translating the SysML Mobile Handset design that is shipped with Rhapsody. The Mobile Handset design contained one external block diagram, three internal block diagrams, three state machine diagrams, two activity diagrams, one parametric diagram, and finally three sequence diagrams.

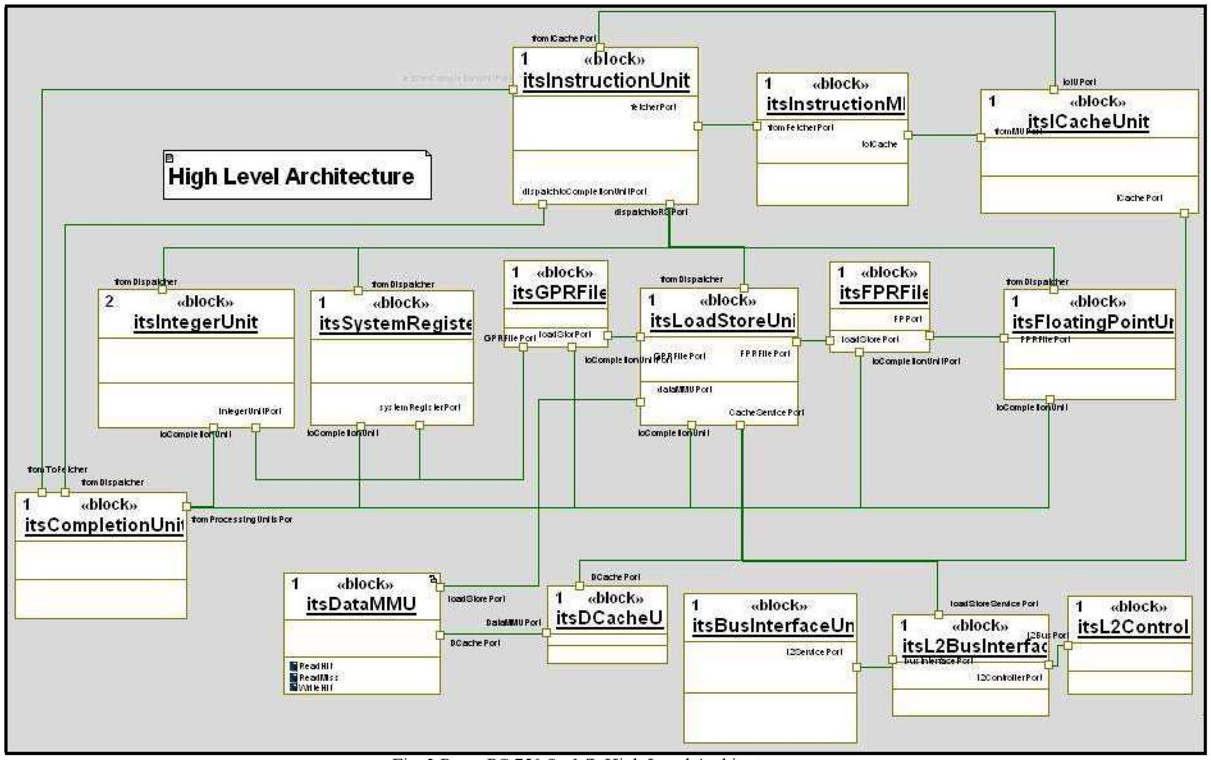

Fig. 2 PowerPC 750 SysML High-Level Architecture

The design has three levels of hierarchy. The translator was able to generate SystemC code that behaved as designed in the SysML notations. The generated SystemC code was able to compare the dynamic sequence of events, produced due to the design dynamic description, with the set of events described in the sequence diagrams. To test the translation and validation process, we have introduced intentional errors in the design dynamic description in the SysML description. The generated SystemC executable was able to report the intentional errors when executed.

We have published the results of translating this design in [13]. We have also used the translator to generate the SystemC code of another SysML design, a radio tuner design. The translation process was successful and gave similar results to the Mobile Handset experiment.

We used the SysML to SystemC translator to generate the SystemC code of a larger and more complex design: the PowerPC 750 design. We have started the SysML design from scratch using the information available in the PowerPC 750 user's manual. We have designed the SysML external block diagram, shown in figure 2, to be very similar to the PowerPC 750 block diagram specified in its user's manual. The SysML blocks contain up to four levels of hierarchy represented in internal block diagrams to describe the contents of each processor building block. The structural design abstracts the connectivity between different modules into messages passed between them via SysML ports. The exact pin-to-pin connectivity will be modeled when the design goes into the model refinement process during the real implementation step of the design. Model refinement from SysML/SystemC abstraction level is beyond the scope of this research, though it is one of its immediate near plans.

The translator generated a C++ class for each package introduced in the PowerPC SysML design. The translator also generated a SystemC module, SC_MODULE, for each block and part existed in the design. Parts are represented as member variables in their SystemC container. The translator glued the structural SystemC together by creating intermediate channels to make the required connectivity between different modules. Service ports map to SystemC ports, sc_port, with appropriate direction and data type. Attributes and operations of either the SysML package or block map to class or sc_module member variables and methods respectively.

To model the dynamic description of the PowerPC, we have started by the Cache-coherence state-machine diagram that represents the Modified-Exclusive-Invalid cache coherency protocol that PowerPC 750 has adopted. The translator was able to translate the state machine diagram to its SystemC code equivalent in the DataMMU block. The SystemC generated code for the state machine looks like the two-process modeling technique used to model RTL state machines.

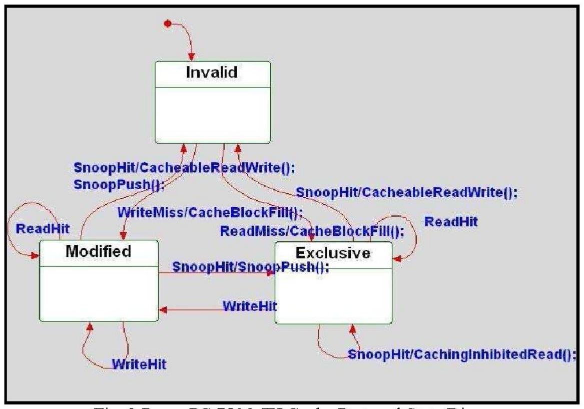
Invalid
SnoopHit/CacheableReadWrite();
SnoopPush();
SnoopHit/CacheableReadWrite();
WriteMiss/CacheBlockFill();
ReadHit
ReadHit
ReadMiss/CacheBlockFill();
Modified
Exclusive
SnoopHit/SnoopPush();
WriteHit
WriteHit
SnoopHit/CachingInhibitedRead();

Fig. 3 PowerPC 750 MEI Cache Protocol State Diagram

Work is on going to create the state machine diagrams of the BusInterfaceUnit and LoadStoreUnit. We are planning to model the integer and floating processing units using activity diagrams. The main hurdles in modeling the dynamics of the system is to select the appropriate level of details that will make the system specifications function without going into the details of the actual implementation.

V. CONCLUSIONS AND FUTURE WORK

In this research, we have highlighted the importance and feasibility of automatic generation of SystemC code from different SysML diagrams. We have presented a way to benefit from the SysML potential ambiguity in validating the overall system integrity. We have presented different ways to generate SystemC code from SysML specifications. In addition, we have presented SysML designs that we managed successfully to translate them into SystemC.

Future work of this research varies between improving SystemC translation technique and leveraging more on the SysML diagrams to validate the whole system at the SysML/SystemC level. The first thing to consider in near future is to make SysML/SystemC translator based on the Eclipse UML2 data-model. This step will make the translator vendor-tool independent. In addition, we still feel that more research work needs to be done to create more sophisticated test benches, at the SysML level, that can be used in system validation at subsequent levels of abstractions. Another near future goal is to compare the generated SystemC code against hand-written code. Comparison can be done via code analysis tools like LogiScope of Ilogix. To perform the comparison, we need to have access to a fair number of hand-written SystemC designs, model them into SysML, and generate SystemC code through our translation tool. Having a linked flow from specification down to implementation was always the dream of system designers, so at the point we feel that system-level synthesis has matured enough, we need to link this work with the down stream synthesis tools.

REFERENCES

[1] Arnout G. and Brophy D. 2004 "SystemC and SystemVerilog Designing SoC Together" Open SystemC Technology Symposium. San Diego, CA. DAC.

[2] Basu A. S., Lajolo M., Prevostini M. 2005 "A Methodology for Bridging the Gap between UML and Codesign", UML for SoC Design Book by Kluwer/Springer.

[3] Butler M., Snook C. 2006 "UML-B: Formal modeling and design aided by UML", ACM Transactions on Software Engineering and Methodology.

[4] Czarnecki K. and Helsen S. 2003 "Classification of Model Transformation Approaches" OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.

[5] Grother T., Martin G., Liao S., Swan S. 2002 "System Design with SystemC". Kluwer Academic Publishers.

[6] Martin G. 2003 "SystemC and the Future of Design Languages: Opportunities for Users and Research" Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI '03), IEEE.

[7] Mellor S., Balcer M. 2003 "Executable UML: A Foundation for Model-Driven Architecture", Addison-Wesley.

[8] Mellor S, Wolfe J., McCausland C. 2005 "Why Systems-on-Chip needs More UML like a Hole in the Head", Design, Automation, and Test Conference in Europe.

[9] Nguyen K. D., Sun Z., Thiagarajan P.S., Wong W. 2004 "Model-driven SoC Design Via Executable UML to SystemC", RTSS'04.

[10] Raslan W., Sameh A. 2007 "Mapping SysML to SystemC", accepted paper in "Forum on specification & Design Languages", FDL 2007, Barcelona, Spain.

[11] Riccobene E., Scandura P., Rosti A., Bocchio S. 2005 "A SoC Design Methodology Involving a UML 2.0 Profile for SystemC", 5th ACM international conference on Embedded software.

[12] Rosenstiel W., Kuhn T., Schweizer T., Winterholer M., Schulz-Key C. 2004 "Object-Oriented Modeling and Synthesis of SystemC Specifications", IEEE.

[13] Sameh A., Raslan W, 2007 "Accelerating SoC Design using SysML and SystemC", ISC, Delft University of Technology, Delft, The Netherlands.

[14] Tan W. H., Thiagarajan P.S., Wong W. F., Zhu Y., Pilakkat S.K. 2004 "Synthesizable SystemC Code from UML Models", RTSS'04.

[15] Zhu Q., Matsuda A., Kuwamura S., Nakata T., Shoji. M. 2002 "An Object-Oriented Design Process for System-on-Chip using UML". In The 15th Int'l Symp. on System Synthesis. ACM.