PROGRESS: PROcess GRaph For Embedded Systems and Software: Combining Top-down and Bottom-up System Design Methodology

James R. Armstrong Shekhar S. Agrawal Hiren D. Patel Sandeep K. Shukla

Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic and State University
Blacksburg, VA 24061
Email: {jra, ssa, hipatel2, shukla}@vt.edu

Abstract

In this short position paper we briefly describe a methodology under development at Virginia Tech, for combining specification oriented design methodologies and IP-reuse based design methodologies for designing embedded systems. Specification oriented paradigms are ***top-down****, and refinement based, and most of them have sound mathematical modeling, abstract representation, and supporting formal methodologies. Ptolemy or SpecC can be cited as examples of such methodologies and frameworks. However, in the industry,* ***cut-and-paste reuse****, and IP* ***component based reuse*** *are used quite often. These reuse methodologies are often* ***ad-hoc*** *in the absence of tools and techniques to create abstract models from the reusable components, and often due to the fact, that in practice part of the design is done in a specification oriented manner, while some parts are taken from existing code base within a company or from purchased IP-cores. We propose using an abstract intermediate representation of component functionalities, namely, Process Model Graphs (PmG). Various manipulative operations on such representations can be viewed us ways to trim, merge, and join IP blocks for systematic reuse. This methodology will allow us to manipulate reusable cores in the abstract, and make top-down specifications merge with PmG abstractions of cores, and then either map them to existing reusable components, or generate required components with tool support. This would lead to a design methodology which successfully combines two different methodologies into one and enables a framework that can benefit from both the approaches.*

I. Introduction:

In the changing world of networked and ubiquitous computing, increasingly sophisticated handheld and bio-medical devices, embedded computing is becoming pervasive and increasingly complex. Furthermore, Moore's Law is imposing an even greater challenge by the continuous improvement in silicon technology at an exponential rate while the design productivity of engineers and computer aided design tools are not increasing at the same pace. This 'Productivity Gap' in semiconductor design industry has necessitated research and development in design and validation methodologies that can enhance productivity, and render validation reliable and inexpensive. A number of approaches have been proposed in this context with the recent developments in system level design languages and models of computation which influence the choices of reuse of intellectual properties and provide efficient simulation and validation environments. However, almost all proposed solutions to the problem of system engineering have addressed one of the two design flows. They have either been *Top-Down specification oriented* using languages such as Ptolemy or SpecC or a *Bottom-up IP reuse approach* using languages such as VHDL and SystemC.

In the Top-Down design flow, the process begins with a specification from which an abstract mathematical model of a system or a portion of a system is constructed. This model is validated to ensure that the specification is interpreted accurately and different algorithms for implementing the system behavior are explored. Several stages of refinement follow this step adding more details to the abstract model. On the other hand, the bottom-up approach exploits design reuse to achieve productivity necessary to build complex systems. Historically, Chip Design companies have always reused designs in going from one product generation to another, but the efficiency of bottom-up design is enhanced by the reuse of IP-cores that a company can buy from an outside source. The challenges with using these IP-cores is designing interfaces between the cores and other logic as well as being confident about their correctness.

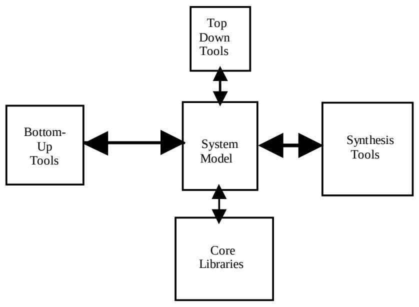

Figure 1. Integrated Top-Down/Bottom-Up Design Approaches

II. Problem Statement

It is a commonly held belief among system designers and CAD practitioners that one-of-a-kind domain-specific construction methodologies for such systems will not be sufficient. Radical advances in systems engineering are needed to allow composition, reasoning and validation of these novel systems and applications. In fact, such combining is practiced in the industry in an unorganized manner in almost every product design cycle. However, lack of formalization of this combined methodology leads to lack of tools and CAD environments, which could lead to better productivity in the design teams.

Figure 1 shows a schematic of our view of industrial design practice, which we believe is an ad-hoc combination of both top-down and bottom-up design styles. The center block shows the system model that integrates the designs from the top-down and bottom-up design flow processes and allows validation of its correctness. For example, when designing a new microprocessor, a top-down model is first built for quick evaluation of performance and feasibility but due to the existence of code and IP-cores in house and off the shelf, eventual implementation is often largely based on existing IP blocks. Unfortunately, in most design cases, this reuse is often cut-and-paste based rather than being based on a solid methodology. Therefore, in our opinion, neither a top-down nor a bottom-up methodology by itself is an adequate design approach, but a combination of the two can lead to new system engineering tools and techniques that will enable System Designers and Engineers to design systems from IP-cores in a specification oriented manner. This approach will allow the development of semi or fully automated refinement methodologies targeting IP-cores or parts of IP-cores to be automatically extracted from IP blocks. However, is it possible to develop a sound and robust methodology and design environment that can combine a top-down specification and models extracted from cores into a system model from which one can map to existing cores, synthesize common protocols between cores and synthesize parts of glue logic?

A comprehensive, robust methodology and environment to combine the different design approaches has not yet emerged due to the lack of a single uniform semantic foundation for the specification of either the high level models or the cores represented at low level. To the best of our knowledge, no work has been done in combining the specification oriented approach with the core reuse based approach because of this lack of uniform semantic foundation. In this paper, we introduce a robust system engineering environment for embedded hardware/software system that brings together the best of both worlds. This environment is currently under development.

III. Related Work:

The approach presented here is based on advances in specification oriented top-down and bottom-up IP-core reuse based hardware and software design methodologies. Academic projects and some CAD/EDA vendors have supplied some tools and techniques, such as Ptolemy [10] and Metropolis projects [9] from UC Berkeley, VCC from Cadence [4], Co-Centric System Studio from Synopsys [16], Rational Rose [11] and other visual tools for software development. These tools and methodologies guide development of complex hardware/software systems through visual programmatic means, which can be viewed as top-down development from high-level specification enabled by a manual or semi-automatic refinement methodology. On the other hand, extensive bottom-up IP reuse [17] either in the form of *cut-and-paste* of existing HDL code or software code, or to some extent by propriety standards for creating IP blocks is occurring in the industry. Evidently, we see a need to combine these two design approaches and construct a comprehensive and robust methodology and environment.

Attempts to construct a robust methodology began with Kahn in 1974 by proposing the use of processes as fundamental to modeling. In 1974, Kahn presented the idea of Process Networks, i.e., sequential processes with blocking read

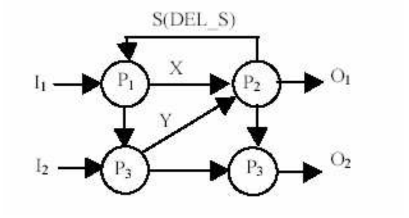

Figure 2. Process Model Graph

interconnected by infinite capacity FIFOs [6]. Process Behavior was implemented using Kahn's Language constructs. VHDL and Verilog employ process mechanisms [5, 12]. SystemC has the notion of processes defined as methods, threads and clocked threads. In [1, 2] we advocated the use of PmGs to represent VHDL behavioral models. Formal properties of processes were developed in [7]. Recently, we performed hardware/software co-design of a GSM system using the PmG representation. [3, 13, 20]

IV. Our Approach

The basis of our semantic foundations for core representation and top-down specification refinement is proposed in the form of ***Process Model Graphs (PmGs),*** and operational semantics of each operation at the tool level are given in the form of operations on PmGs. We provide details of the denotational and operational semantics for PmGs in [22] and limit our paper to a brief overview of PmGs. Firstly, a Process Model Graph (PmG) is a digraph [21] whose nodes represent processes and whose arcs represent signals as shown in Figure 2. Signal arcs are labeled with the signal names they represent, e.g., X and Y. Sometimes the name of the generic delay for a signal is added to the signal name, e.g., S(DEL_S). The PmG represents a module. The input arcs I1 and I2 represent module inputs, and the output arcs O1 and O2 represent module outputs. Other arcs represent internal feed-forward (X and Y) or feedback (S) signal flows.
The operations that can be performed on a PmG are ***join, trim*** and ***merge***. Joining and trimming are most frequently employed when IP-cores are ported into the System Model PmG. Cores are either hard or soft cores. For ASICs, hard cores consist of the IC layout and for FPGAs the bit file. Soft cores consist of an RTL description in VHDL or Verilog. Hard cores must be accompanied by behavioral models so the core can be joined with other models for simulation purposes. Soft cores may not have behavioral models because the RTL model can be used for this purpose. IP-cores are more marketable if they offer a range of capabilities. However, when constructing the PmG system model we wish to use specific instances of the core. Thus, before joining the core model to the PmG, the core must be trimmed of redundant behaviors. Below we discuss the nature of the trim and join operators. We present these concepts in the context of an example where core models are to be inserted into a system model implemented via SystemC.

Figure 3 shows a core based model which interfaces a FIR filter to a two channel CODEC. There are two cores in the model: 1) the FIR filter is a hardcore which is produced by Xilinx's Coregen Program [19] 2) the Clock Generator and the Channel are soft VHDL cores provided by the board manufacturer [18]. The CODEC is fixed hardware and the core based model is a VHDL model to be implemented in FPGAs.

Trim Operation: The trimming operation generally results to three forms of elimination of redundancy in the existing cores. For example, one might have an ALU core, that does both integer, and floating point computation. If the required use, does not require floating point operation, one can use program slicing techniques to eliminate the parts relevant to floating point operations. In the example of Figure 3, we have used three kinds of trimmings as follows:
Functional Elimination: three processes are totally eliminated because high level knowledge of the model shows that these processes are not required.
Eliminated By Code Trimming: instantiation of the parameter values trims the code of process such that the process merely passes inputs to outputs without modification and without delay, resulting in the elimination of that entire process.
Internal Code Trimmed: instantiation of the parameter values in trims the code of process by a certain percentage.
For example, The FIR filter is a Serial Distributed Arithmetic FIR Filter. It performs the

required multiply – accumulate operations serially to save chip area. The core can either be used by itself or a number of them cascaded together. It can be specialized in a number of ways depending on the nature of the FIR filter being designed. Accompanying the core is a multi-process, VHDL behavioral model. Applying the Trimming operation for this core, it is found that a total of 10 functions are eliminated and the resulting core had 39% less inline code than the original core.

Join Operation: As shown in Figure 3, an Interface Module is used connect the CODEC

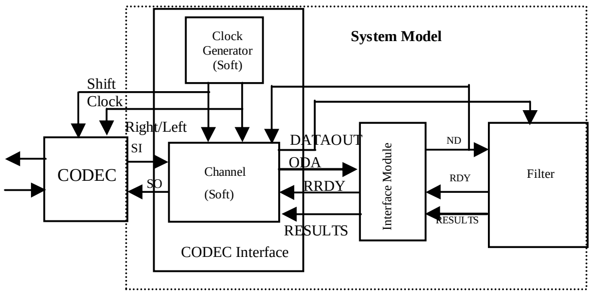

Figure 3. IP-core system model

Interface Core to the FIR core. It performs two functions: a) converts an Output Data Available (ODAV) level to a ND single clock period pulse, and b) registers the filter results output which is stable only for one clock time. Implementation of this interface module is an example of a *join*. In order to automate the joining process, one may have to use *behavioral typing* [8] of core interface signals. In the system illustrated in Figure 3 two data types are used: 1) for control: std_logic, and 2) data: std_logic_vector. As part of an automation process, one can define two behavioral types: *control* and *data*. The join operation consists of type conversion between behavioral types, e.g., the Interface Module in Figure 3 performs two behavioral type conversions: a) type 1_level to type p_pulse and b) type momentary to type registered. Certain assumptions underlie this type hierarchy, i.e., pulses last for one clock cycle as does momentary data.

Merge Operation: Often two cores may be merged in the context of eliminating redundant signals [3].. External signal storage is replaced by internal variables. The read/update mechanism for variables is designed to ensure the same computation.

Mapping Operation: A process whereby two different PmGs in distinct domains can be related to each other by certain mapping definitions provided in [22]. Rigorous definition of mapping is needed to automate the process of mapping a PmG to PmG models of existing modules.

Example

To test our PmG structure and the operations associated with it, we developed a System PmG and an associated SystemC model for the system shown in Figure 4 which is an example for PmG mapping. We utilize the cores described in Figure 3. The CODEC is a two channel, 20 bit A/D, D/A converter, with serial input and output. The CODEC Interface performs a serial-to-parallel (read) and parallel-to-serial (write) conversion. In the system shown in Figure 4, only the input from the CODEC on one channel is used. Thus, the CODEC Interface core is trimmed to a single channel, read only configuration. The CODEC Interface Core is RTL VHDL. The hardcore FIR filter is modeled by behavioral VHDL and is also trimmed to match the specifications. The join operator is in VHDL. The BUS is a high speed serial bus implemented in SystemC, the FFT and Format and Display actors are Ptolemy actors operating in the SDF domain.

We translated the above mapping example into a SystemC model as shown in Figure 5. The experiment validated the usefulness of our PmG operations. It was conducted manually, but automation of this process is under investigation.

Our PmG approach not only provides a robust and comprehensive methodology for design but the integration of hardware/software co-design is a direct byproduct of the proposed methodology. It is very important that modeling, hardware/software co-design, and synthesis be integrated. To put it in a nutshell, the system procedure can be split into 5 different steps as:

1. The Process Model Graph (PmG) is extracted from the SystemC system level model.
2. Each Process P_i is characterized in terms of its software (C_{soft}) and hardware (C_{hard}) complexity. An Instruction Set Simulator (ISS) or profiler for the core processor is used to determine C_{soft} for

each P_i whereas a Behavioral Synthesis Tool determines C_{hard}.

3. Using the process complexity values from the Process Library and deadlines from the Constraint Library, the Graph Partitioning Algorithm partitions the graph into hardware and software sub-graphs
4. In step 3: the Graph Partitioning Algorithm generates a new set of process complexities for those processes assigned to the hardware sub graphs. Also, whenever a signal crosses the boundary between the hardware and software sub-graphs, delay overhead is incurred because of bus delay and processor task switching time. This information is back annotated to the PmG where new delay values are assigned to nodes in the hardware sub-graph and process nodes are added to the graph at the partition boundary to account for bus and processor task switching delays.
5. The Graph Partitioning Algorithm then rechecks the deadlines.

This approach was applied successfully to hardware software co-design of a system which implements the GSM transmitter and receiver sections. The software target was the StarCore 140 Processor; the hardware targets the Synopsys SystemC compiler. The system automatically identified partitions between hardware and software that meet all GSM timing constraints [3].

Case Study

Ptolemy is a top-down design based tool that supports heterogeneous modeling, simulation, and design of concurrent systems [10]. Different Models of Computations (MOCs) that deal with concurrency and time can be modeled using this tool. Each MOC gives an interaction mechanism for components in the model. In PtolemyII, each MOC is represented in a different *domain*. The choice of the type of Director dictates which particular Model of Computation is to be used. The chosen Model of Computation provides execution semantics for actor components of the model and the method of communication between actors. The model of communication is implemented in the Actor receiver, which is specific to the Model of Computation. In SystemC, the execution semantics is implemented by the simulation kernel. The method of communication is chosen by the way in which the Channel between actors is implemented. In Ptolemy, the receiver is buried inside an actor, thus communication is implicit in the Model of Computation. In SystemC the Channel is an explicit component. The SystemC kernel implements a two phase simulation cycle: 1) EVALUATE PHASE: Evaluate the set of currently triggered processes 2) UPDATE PHASE: Update Channels to values within the same evaluations cycle (zero time delay), after delta delay, i.e., updated in the next update cycle and results are available at the next evaluation cycle, or updated at a later time in the simulation during an update cycle. During the evaluation phase, processes notify events and during the update cycle those events are triggered. These Ptolemy Actors can be coded in a behavioral fashion and modeled as PmGs. We have conducted work in [22] where we provide an example of an automation of transforming a Ptolemy Actor into a SystemC module. Current work in constructing PmGs from SystemC modules is also underway, allowing for a rendez-vous for Ptolemy Actors to be brought down to the PmG level and IP-cores to be brought up to PmG level given that IP-cores were provided in SystemC modules. Once having constructed the PmG, the above described operations can be performed on the PmG. This trimmed, joined and merged PmG can then be then mapped onto an IP-core for future industrial reuse.

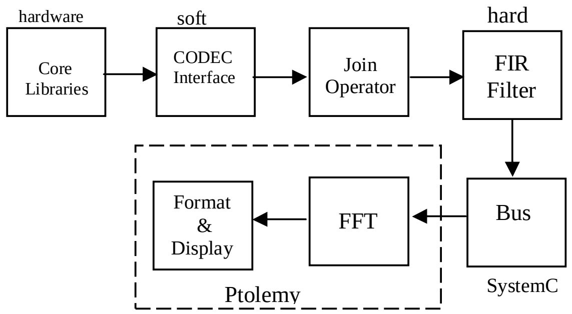

Figure 4. Comprehensive PmG Mapping Example

V. *Future Work:*

Automation of this new methodology will be the key to faster and better reuse of cores, together with merging top-down and bottom-up methodologies. Our focus is to automate the extraction of PmG models from SystemC cores, implement a framework to apply operators, namely, Trim, Join and Merge, followed by creating a robust design framework that facilitates design reuse. Development of a GUI for easy user interaction and reduced design time is also one of the long term goals of our work. We also plan on adding validation capabilities in the framework.

VI. Conclusion:

In this paper, we have introduced a new development methodology that brings together two different design flows, the top-down and bottom-up flows. The methodology makes use of Process Model Graphs (PmGs) to achieve this goal. The various operations that PmGs support, namely Trim, Join and Merge and the ways in which these operations help in developing the model has been shown. The entire design methodology is illustrated by a comprehensive example that makes use of all the design aspects. In summary, PmG based representation of system models as well as core models and automatic extraction and manipulation of such representation will enable us to create an environment and methodology to combine a core-reuse based methodology with specification driven methodology.

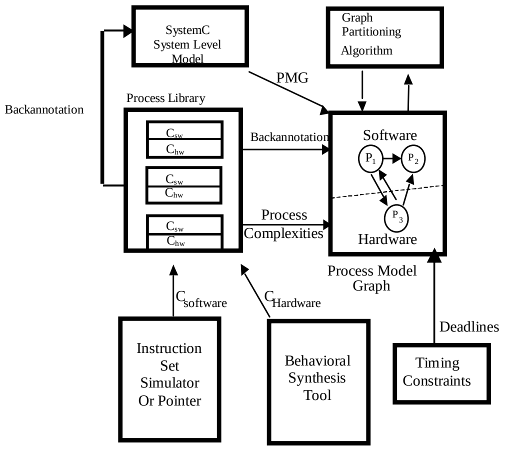

Figure 5. PmG Based Partitioning, and Synthesis System

VII. References:

[1] J. R. Armstrong and F. G. Gray, VHDL Design Representation and Synthesis, Englewood Cliffs, N.J., Prentice Hall, 2000.

[2] J. R. Armstrong, Chip Level Modeling With VHDL, Prentice Hall, June 1988,

[3] J.R. Armstrong, J. M. Baker, and P. Adipathi, “Model and Synthesis Directed Task Assignment for Systems On A Chip,” Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems, Louisville, KY, September 2002

[4] Cadence Website on VCC, http://www.cadence.com/products/vcc.html

[5] IEEE Standard 1076-93 Language Reference Manual.

[6] G. Kahn, "The Semantics of a Simple Language For Parallel Programming", Proceedings of the IFIP Congress 74, 1974, North Holland.

[7] A. Lee and A. Sangiovanni-Vincentelli, " A Framework for Comparing Models of Computation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17. No 12. December 1998, pp 1217-1229.

[8] Edward A. Lee and Yuhong Xiong, "Behavioral Types for Component-Based Design," Memorandum UCB/ERL M02/29, University of California, Berkeley, CA 94720, USA, September 27, 2002.

[9] Metropolis Project Home page, http://www.gigascale.org/metropolis/

[10] “Ptolemy II, Heterogeneous Current Modeling and Design In JAVA,” Memorandum UCB?ERL M)1/12, March 15,20001

[11] Rational Rose Webpage, http://www.rational.com/products/rose/index.jsp

[12] Sagdeo, V, The Complete Verilog, Book, Kluwer Academic Publishers, Boston, MA, March 1998,

[13] B. Sirpatil, J. M. Baker, J.R. Armstrong, “Using SystemC to Implement Embeded Software,” Proceedings of HDL 2002, San Jose, March 2002, pp 41-45.

[14] J. E. Stoy, Denotational Semantics: The Scott-Strachy Approach to Programming Language Theory, Cambridge, MA: MIT Press, 1977.

[15] SystemC Consortium: www.systemc.org

[16] Synopsys Co-Centric Studio product webpage, http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html

[17] Virtual Socket Interface Alliance Webpage, http://www.vsi.org

[18] www.xess.com

[19] Xilinx , www.xilinx.com

[20] A. Varma, J. R. Armstrong, and J.M. Baker, “A SystemC GSM Model for Hardware/Software Co-design,” Proceedings of HDL 2002, San Jose, March 2002, pp 41-45.

[21] Wilson, Introduction to Graph Theory, Academic Press, NY, 1972.

[22] J. R. Armstrong, and S. Agrawal, “Denotational and Operational Semantics of Process Model Graphs,” Submitted. Available at http://www.visc.vt.edu/armstrong/jra_vt_pmg.pdf