

## Experience in using several semantics to design DSP systems with VHDL : an architectural view

Xavier Warzee  
Thomson-CSF Technologies & Methods  
warzee@ttm.thomson.fr

Philippe Kajfasz  
Thomson-CSF Optronique  
philippe.kajfasz@thomson-csf.fr

### Abstract

*We present our approach to the definition of a multi-paradigm environment to specify, model and synthesize embedded DSP systems. Instead of assuming one semantic ( for instance, synchronous semantics [15] ) to specify and model the whole system, we propose to use simultaneously several semantics to describe and validate key system properties. Traceability of system requirements is challenging with such an approach. We propose an architectural view of systems which defines components and connectors to support respectively functional and non-functional requirements modeled with specific semantics.*

*Using the object-oriented framework Ptolemy [13] , which supports simulating and prototyping of heterogeneous systems, a global view of the system is ensured, allowing us to validate its behavior.*

### 1. Introduction

Building embedded Digital Signal Processing (DSP) systems means concurrent development of hardware and software components since these components may collaborate to satisfy functional and non-functional requirements. Functional requirements specify functions (algorithms and data) supported by systems. Non-functional requirements specify constraints (time, space, cost, etc) applied to systems.

VHDL lets us specify hardware systems from the functional to the hardware levels. VHDL specifications are executable and used for simulation and synthesis with EDA tools. Thus, not only the VHDL syntax is important, but also its semantic. A semantic approach studies programs and their executions as mathematical objects. Thus, it is

possible to study the nature of program computations and the rigorous deductive reasonings that can be applied to prove their run-time properties. Specific languages or formalisms, with precise underlying semantics, can not only specify systems statically but also their run-time properties. We may then execute those specifications for tests, and apply formal validation. For instance, telecommunication systems may be specified with the SDL language because its semantics are well-defined for this domain of applications. Identifying semantics really needed to specify all the properties of systems not only allows us to ensure satisfaction of requirements, but also to explore the real nature of the systems.

As the complexity of systems grows to meet expanding requirements in terms of performance and services with the constraints of shrinking budgets and schedules, the integration of COTS and standardized components becomes more important. An architectural view [18] supports systems description at the components level, and components inter-actions, with constraints and rationale. A mapping between requirements and architectural elements provides a first model, which may be refined at each iteration of the development process (Boehm Spiral model of development). Hierarchical composition of components lets us define layers [2] providing virtual machines for building systems. Virtual machines provide an abstract view of sub-systems which hides architectural trade-offs and design choices. A layered architecture enforces modularity and eases system evolutions by hiding from the upper layers the integration of external components such as COTS which may change independently of the systems' life. Component composition also enforces adaptability of system development when requirements evolve. For example, a controller may be added to DSP

algorithms to supervise their sequence of execution according to external events.

## 2. Properties of embedded DSP systems

For DSP systems, the dataflow model of computation [6] is often used. The semantics of this model define fixed functions on data streams (no control structures like the while-loop). Input data arrive continuously and are processed through the DSP system to provide for example plots for high level algorithms. For radar and sonar DSP applications, the dataflow model is well suited. But for embedded systems like multi-mode radars in the defense industry, we need a deterministic model which ensures reaction of the system to external events at a speed imposed by the environment. The synchronous approach [3] which allows us to operate explicitly on events provides a general framework to model reactive systems. In order to validate fully DSP systems, these models of computation must be composable to study all inter-actions in the systems and with their environment.

## 3. Semantics based integration

We used Ptolemy as a backplane for tool and model integration for DSP system simulation and prototyping. The architecture of Ptolemy allows simulations and prototyping of heterogeneous systems based on models of computation with their own semantics.

The Discrete Events (DE) model is often used to simulate VLSI chips, communication networks or transportation systems. VHDL simulators are based on the Discrete Events model. But DE model implies a total order among events in the modeled system. This total order leads to overspecified systems when total ordered events are not necessary. Models of computation based on partial ordered sets of events [5] may help to express systems properties, such as concurrency [16], and provide a formal framework [19] for validation. Thus, it is interesting to use the Static DataFlow [11] (SDF) model which is well-suited for DSP applications. VHDL code may be generated from an application modeled with an SDF approach to obtain, for instance, efficient code for simulation or the VHDL code supporting DSP algorithms in an embedded system.

We have identified several models in Ptolemy to support formal specifications of our DSP systems and have chosen :

- SDF (DataFlow) for functional requirements, and
- SR (Synchronous/Reactive close to the Esterel [4] semantics available in release 0.7 of Ptolemy) for systems control.

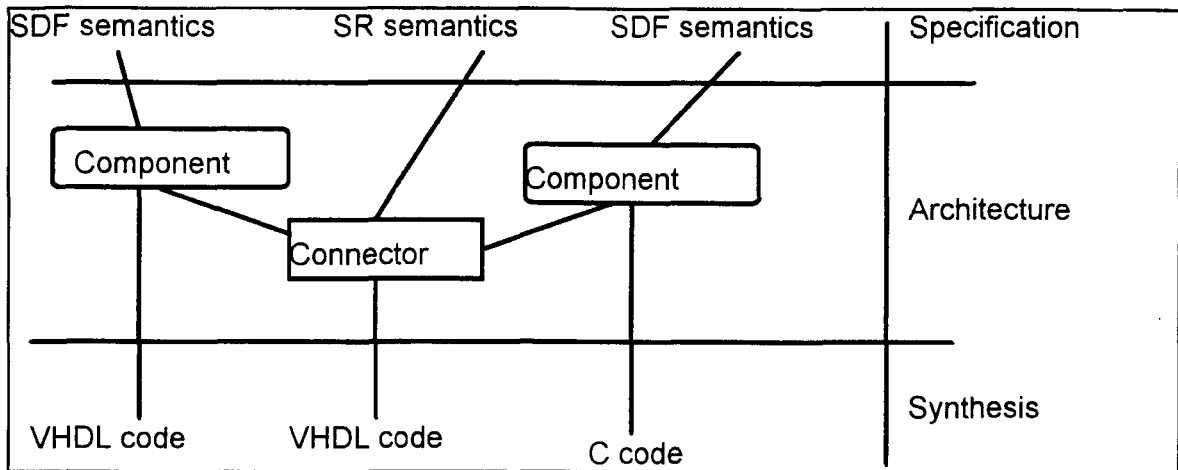
Composing specifications expressed with different formalisms implies studies on semantic interactions. One of the aims of the Ptolemy project is to explore semantic interactions to provide an integrated environment. Using simultaneously various models allows better traceability between requirements and system models. Orthogonal requirements such as performance and functional requirements remain orthogonal in system models but can be composed hierarchically.

## 4. Architectural view

An architectural approach defines several architectural styles which describe collaboration paradigms between components but with constraints at the system level. Architecture styles are defined through the notion of connectors which define how components collaborate. Typically, non-functional requirements, such as concurrency, synchronization and real-time constraints, are supported by connectors, and functional requirements by components. Hierarchically composition of components allows us to refine component and connector descriptions in terms of finer components/connectors. Thus, it is possible to describe an architecture, which supports functions and services, and satisfies constraints with trade-offs without losing strong links between requirements and the architectural elements supporting them.

In our approach, we used semantics to specify and model DSP systems, and then to generate VHDL code. These semantics allow early validation of system specifications. These semantics provided underlying models of collaboration at the architecture level (dataflow, reactive) for components and connectors through the SDF and SR Ptolemy domains. During the code generation, VHDL codes are generated for simulation and synthesis which follow the semantics used.

VHDL subparts, generated from SDF descriptions, of a system may be validated with the same C testbenches used for algorithms validation. With Esterel, formal validation allows us to generate directly proven VHDL codes for the control part of a system.



**Figure 1**

We may use in the future tools such as RDD-100<sup>TM</sup> to capture informal requirements and enforce traceability from system specifications until hardware/software elements by successive refinements and their allocation to system architecture.

The architectural approach helps to identify functional and performance requirements. An early identification of these types of requirements may ease mapping with formal descriptions using dedicated languages (Esterel, SDL, VHDL) or models (CCS [14], CSP, SDF).

This approach enforces semantics based partitioning of system specifications. The resulting partitions are less subject to changes because they represent fundamental properties of the systems.

Strong models or formal languages provide their own framework to explicit properties and to validate/simulate them. Intermediate formats provide an abstract representation which helps to keep independent specification partitioning from system synthesis. For instance, process calculus applied to the BLIF code generated from an SR description may result in a different partitioning than in the SR representation.

## 5. DSP system design experience

To validate this approach, we made some experiments based on these semantics. Using a top-down methodology, we first specified and simulated a DSP system at system level taking into account DE, SR, and SDF models. For each state in SR, a dataflow graph can be attached and at the opposite, an SR controller may be embedded in a dataflow graph. The DE model provides a global model of the system architecture (including protocol modeling).

Next step consisted in automatic code generation for specific targets. In that case, the chosen target defines the architecture topology; types of communication links; generated language; compilers, linkers, loaders used; optimizations (memory size, communications overlapping for parallel targets).

From SDF and SR models, C and VHDL codes may be generated either directly for SR models, using for example a « synchronous » language such as Esterel which can produce synthesizable VHDL code, or through specific interfaces with CAD tools like Leapfrog<sup>TM</sup> for simulation and SYNERGY<sup>TM</sup> for synthesis.

The main advantage of using Esterel is that we were able to do formal verification with tools such as Tempest from Bell Labs, and XEVE from INRIA/CMA.

In a final step, the generated code corresponding to the whole DSP system can be executed. The prototype we realized allows testing of properties such as concurrency and communication, etc.

Each part of the system description is retargetable which means that the function performed by a component is independent of the computation model. We may use, for instance, a FFT to specify the system with the SDF model and generate C or VHDL codes. The functional parts, like a FFT, can be reuse within an SR model or in a CSP [8] model to simulate process interactions. This is a key concept to ease component integrations and reuse for codesign [10].

## 5. Conclusion

In our work, we considered the different semantics which may be used to model embedded

DSP systems. These semantics allow us to apply formal validation and execution to explore the real nature of run-time properties of the systems. Then, based on an architectural approach, we composed specifications and assigned them to architectural elements such as components and connectors to ensure better traceability between requirements and design choices. Finally, we decided to generate from these architectural descriptions VHDL or C code depending on costs and physical constraints.

## 6. Future work

Formal models for composing specifications like SPECWARE [17] exist which may simplify, in the future, systems validation and generation.

There is also work to do on co-synthesis based on the OMF future standard to interface VHDL and C code and execute them together. Ptolemy offers a practical framework, which provides powerful integration mechanisms, to test our method.

## References

- [1] Gregory Abowd, Robert Allen and David Garlan, « Formalizing Style to Understand Descriptions of Software Architecture », *CMU-CS-95-111*.
- [2] Don Batory and Bart J. Geraci, « Composition Validation and Subjectivity in GenVoca Generators », *IEEE Transactions on Software Engineering, special section on Software Reuse*.
- [3] Benveniste and G. Berry, « The Synchronous Approach to Reactive and Real-Time Systems », *Proceedings of the IEEE*, Vol. 79, No. 9, pp. 1270-1282, 1991.
- [4] Berry and G. Gonthier, « The Esterel Synchronous Programming Language : Design, Semantics, Implementation », *Science of Computer Programming*, Vol. 19, No. 2, November 1992.
- [5] B. A. Davey and H. A. Priestley, « *Introduction to Lattices and Order* », Cambridge University Press, 1990.
- [6] J. B. Dennis, « First Version Data Flow Procedure Language », *Technical Memo MAC TM61*, May, 1975, MIT Laboratory for Computer Science.
- [7] Harel, « Statecharts : A Visual Formalism for Complex Systems », *Science of Computer Programming*, Vol. 8, No. 3, June 1987.
- [8] Hoare, « Communicating Sequential Processes », *Communications of the ACM*, Vol. 21, No. 8, August 1978.
- [9] Garlan, M. Shaw, « An Introduction to Software Architecture », *Advances In Software Engineering*, Vol. 1, World Scientific Publishing Company, 1993.
- [10] A. Kalavade and E. A. Lee, « Manifestations of heterogeneity in Hardware/software Codesign », *Proceedings of DAC-94*, San Diego, CA, June 1994.
- [11] E. A. Lee and D. G. Messerschmitt, « Synchronous Data Flow », *IEEE Proceedings*, September, 1987.
- [12] P. Lehoczky, L. Sha, and Y. Ding, « The rate monotonic scheduling algorithm: Exact characterization and average case behavior », *IEEE Real-Time Systems Symposium*, pp. 166-171, December 1989.
- [13] T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, « Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems », *International Journal on Computer Simulation*, Vol. 4, pp. 155-182, April 1994.
- [14] R. Milner, « *Communication and Concurrency* », Prentice Hall, Englewood Cliffs, NJ, 1989.
- [15] Mooney III, C. Coelho Jr, T. Sakamoto, « Synthesis From Mixed Specifications », *Proceedings of EURO-DAC'96 with EURO-VHDL'96*, 1996.
- [16] R. Pratt, « Modeling Concurrency with Partial Orders », *International Journal of Parallel Programming*, Vol. 15, No. 1, pp. 33-71, Feb. 1986.
- [17] Y. V. Srinivas and R. Jüllig, « SPECWARE<sup>TM</sup> : Formal Support for Composing Software », *Proceedings of the Conference on Mathematics of Program Construction*, July 1995.
- [18] Svoboda, « The Three «R's» of Mature System Development : Reuse, Reengineering, and Architecture », *Proceedings of the Fifth Systems Reengineering Technology Workshop*, Monterey, CA, February 1995.
- [19] Winskel, *The Formal Semantics of Programming Languages*, the MIT Press, Cambridge, MA, USA, 1993.