# RASSP VHDL Modeling Terminology and Taxonomy - Revision 1.0

Carl Hein
Lockheed Martin
Advanced Technology Laboratories
Camden, NJ 08102

Todd Carpenter
Honeywell Technology Center
MPLS, MN 55418-1006

Paul Kalutkiewicz
Lockheed Sanders
Advanced Engineering & Technology
Nashua, NH 03061-0868

Vijay Madisetti
School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250

http://rassp.scra.org

**Abstract:**

*VHDL modeling taxonomy and terminology conventions are emerging from the on-going efforts of the Terminology Working Group (TWG). Based upon examination and comparison of previously published modeling taxonomies, the working group is evolving a multi-axis taxonomy designed to describe the information content of RASSP model types and abstraction levels and to facilitate selection and construction of interoperable models. The TWG used the taxonomy to concisely refine modeling terms applying to system, hardware, and software models; abstraction levels; structural hierarchies; and modeling paradigms. The refined definitions for several of the modeling terms especially important in RASSP are listed and discussed.*

## 1. Introduction

There has been much confusion over terminology among the RASSP organizations. Some organizations use many common modeling terms with divergent meanings, while others use different words to describe the same type of models. Clearly communicating ideas about modeling techniques and model types among organizations is essential to achieving the goals of RASSP. Without a common language, the RASSP community cannot effectively communicate, and the simulation models will be incompatible.

The Terminology Working Group (TWG) was formed at the January 10, 1995 Principal Investigators meeting in Atlanta, GA., to address the modeling and terminology challenge. The core working group consists of five members representing the two prime contractors, a technology base developer, the educator facilitator, and the government.

The TWG's mission is to develop a systematic basis for defining VHDL model types and to use this basis for concisely and unambiguously defining a terminology that describes the models that are used within a RASSP design process. One crucial requirement for the basic taxonomy is that it must be useful for selecting, using, and building appropriate interoperable models for specific roles in a RASSP design process. The terminology is based on the commonly documented and applied vocabulary in the digital electronic design and modeling industry, and it draws heavily from related previous and ongoing efforts by the EIA [1], ESA [2], Army [3], Navy [4], and from the annals of related literature from Design Automation Conference (DAC), VHDL International User's Forum [5], and text books[6]. Previous efforts focused on narrower domains than RASSP. RASSP spans many domains, including: parallel processing; multi-board and multi-chassis systems; software; digital signal processing; and application functions, with strong interaction with other domains such as analog, mechanical, physical, and RF [7].

The TWG modified and augmented the previously defined terminology sets. The group broadened parochial definitions, distinguished overlapping definitions, equated close synonyms, removed non-applicable terms, added needed or missing terms, clarified poorly defined or misunderstood terms, and suggested new terms as replacements or synonyms to outdated terms. When appropriate existing definitions were not available for significant terms used within the RASSP community, the TWG attempted to create them.

The refined nomenclature will identify the models developed within the RASSP process that

document design information and the practices for developing interoperable versions of those models. The RASSP program needs to develop broad consensus for a modeling vocabulary that is readily adopted by engineers and students. The vocabulary should promote the requirements of the RASSP process for interoperable VHDL models.

## 2 Modeling Taxonomies:

### 2.1 Existing Taxonomies

The working group initially compared three existing model definition approaches (shown in Table I). The group considered the features of the existing approaches as a foundation for a VHDL model taxonomy[5,8]. The Ecker and Madisetti spaces share two axes, while their remaining axes do not directly correspond. Both have an axis for *Time* resolution and a second axis representing the resolution of data *Values* in a model. Ecker calls the second axis *Value*, while Madisetti calls it *Format*. The Y-chart has a similar axis called *Functional-Representation*. The third axis of the Ecker cube is similar to the *Structural-Representation* axis of the Y-chart but has no corresponding axis in the Madisetti case. (The later situation arises intentionally.)

None of the remaining axes of the taxonomies directly correspond. The Y-chart seems limited to only the logic level. None of the taxonomies appear to directly address the hardware/software codesign aspect.

### 2.2 New Taxonomy

The TWG proposes a new taxonomy that more clearly represents the model attributes relevant to a RASSP designer. The new taxonomy shares characteristics of the earlier approaches, but seeks to expand the representation, as well as use terminology readily understood and used by designers. The proposed taxonomy consists of a common set of attributes to independently describe a model's internal and external resolution. Distinguishing between the internal and external views is important in selecting, using, and building models because it enables clarity and precision. Existing terminology often mixes attributes, as viewed from inside a model, from similar attributes, as viewed from the model's interface boundary.

The proposed axes, like their predecessors, explicitly characterize a model's relative "resolution of details" for important model details. This proposed taxonomy identifies four orthogonal model characteristics:

1) Temporal detail
2) Data Value detail
3) Functional detail
4) Structural detail

Because each aspect is specified from both an internal and external viewpoint, the proposed taxonomy effectively contains eight attributes describing a model's descriptive level: internal temporal, value, structure, and function resolutions, and external temporal, value, structure, and function resolutions. This set of eight attributes does not address the hardware/software codesign aspect of a model, because it does not describe how a hardware model appears to software. Therefore, the set is augmented with a ninth axis (shown in figure 1) to represent the level of software programmability of a hardware model or, conversely, the abstraction level of a software component in terms of the complementary hardware model that will interpret it.

Although section 3 defines the vocabulary terms graphically to show their applicable coverage, a convenient method for specifying a particular model's information content is to use the *Internal/External(temporal,value,function,structure)* notation. For example, the content of a particular algorithm model can be specified as:

**Table I - Comparison of prior and proposed taxonomy concepts**

| Source - Taxonomy | Axes | | | | | | |
|---|---|---|---|---|---|---|---|
| Gajski and Kuhn: Y-chart | | | Struct. Rep. | Funct. Rep. | Geom. Rep. | | |
| Ecker: Ecker cube | Timing | Value | View | | | | |
| Madisetti Taxonomy | Timing | Format | | | | Value | State |
| RASSP TWG Taxonomy | Timing Res. | Data Value | Struct. Res. | Funct. Res. | | | Intern/ Extern |

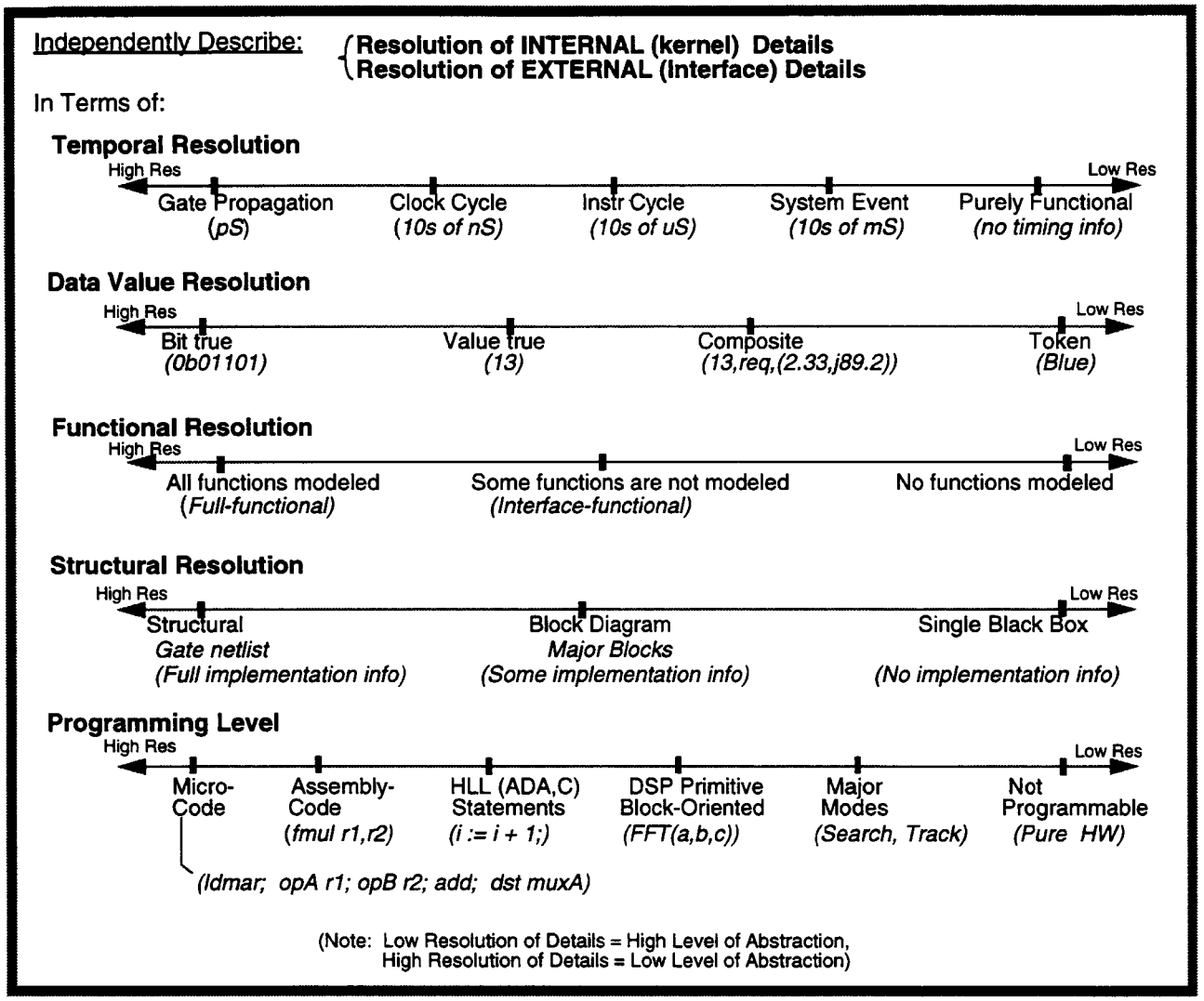



**Figure 1 - New Taxonomy Axes**

*Internal(none,composite,full-functional,none), External(none,composite,none,none), SW-Program(none)*

### 2.2.1 Temporal Resolution Axis

The Temporal Resolution axis represents the resolution of events that are modeled in terms of their time scale. Resolution is analogous to precision, as distinguished from accuracy. For instance, a model's time resolution may be stated in terms of the starting and ending times of major system functions, where each function spans thousands of clock-cycles. In such a case, the model resolves events down to the major function level and not down to the clock-cycle level, even though the accuracy of the starting and stopping times may be specified accurately to within a clock-cycle.

### 2.2.2 Data Value Resolution Axis

The Data Value Resolution axis represents the resolution with which values are specified in a model. Again, note that resolution is analogous to precision, as distinguished from accuracy. For instance, a register containing the value negative-one (-1) may be modeled with high resolution in terms of its actual two's-complement binary "0b111" form, it may be modeled more abstractly as a signed integer "-1", or an even more abstract enumerated type such as "Blue". Each representation could be considered equally accurate. However, the first case resolves the value closer to the form actually contained in the target device. The more abstract the representation of a value is, the less implementation details are resolved.

### 2.2.3 Functional Resolution Axis

The Functional Resolution axis refers to the level of detail in which a model describes the functionality of a component or system. For instance, a highly abstract, low-resolution model could specify the function of a digital filter in terms of its mathematical transformation, while a high-resolution model could resolve the function in terms of the Boolean operations which implement the target device. Both models can be functionally accurate.

In the extreme, the most abstract (or low-resolution) model could contain no functionality at all. In contrast to internal functionality, the external functionality specifies the interface behavior of a device's (or system's) ports.

### 2.2.4 Structural Resolution Axis

The Structural Resolution refers to the level of information detail a model provides about how the modeled component is constructed out of constituent parts. For example, one model of a processor chip may have no information about its internal structure. A second model of the same chip may specify its structure in terms of five major blocks. A most detailed model may specify the internal structure in terms of the interconnection of specific logic gates.

Although more abstract, the second model is perfectly accurate as long as the five major blocks can be identified as connected in the gate-level model. This understanding of structural resolution holds for both external structure and internal structure, as described in the example above. An example of external structure is a component model in which the component's memory-bus port is modeled as a single composite value containing many fields, without describing the port's physical structure. Alternately, a higher resolution model would specify the ports's structure in terms of bit-widths; address and data buses; and handshaking lines.

### 2.2.5 Software Programming Resolution Axis

The Software Programming Resolution axis is the granularity level of software instructions that the model of a hardware component interprets in executing target software. For instance, a network performance model only interprets instructions on the level of Data Flow Graph (DFG) primitives, such as matrix invert, vector multiply, or Fourier transform. Such primitives represent hundreds of lines of source code, but are interpreted as a single instruction in terms of a time-delay by a network performance model. An Instruction-Set-Architecture (ISA) model interprets individual assembly instructions. In this sense, the ISA model is programmable at a much finer granularity, or higher resolution, than the network performance model.

At the lower extreme, a model of a microcode programmable processor is programmable at an even lower level of granularity than the ISA model because it allows control of individual-register and multiplexor structures within the device during execution of an assembly-level instruction. Software design components or non-programmable models are at the opposite extreme because neither interprets programmable instructions.

## 2.3 Tentative Additional Attributes

In addition to resolution (precision), the TWG is considering the inclusion of additional attributes. For instance, the Temporal and Data Value axes could specify an accuracy aspect as a percent tolerance and whether a model describes actual, minimum, typical, or maximum values. A "completeness" aspect is also being considered that would specify, for example, the portion of functionality or particular functions that the model describes or excludes from the model. The resolution, accuracy, and completeness aspects would accompany the axes in the same way the internal/external aspect does.

# 3 Vocabulary

In Table 2, the working group's initial modeling terms are categorized according to major area, such as: abstraction levels, system levels, hardware levels, software levels, structural hierarchy levels, and general modeling-type terms.

Following are selected terms as currently defined by the TWG for general modeling terms, system, architectural, software, and hardware models respectively. To avoid vague or circular definitions, the group placed emphasis on providing examples to accompany the definitions. These examples should provide a level of understanding and concreteness to any discussions regarding the terms. The examples also tend to tie the terms to their intended uses and domains. To reduce the tendency of examples to limit the definitions, the

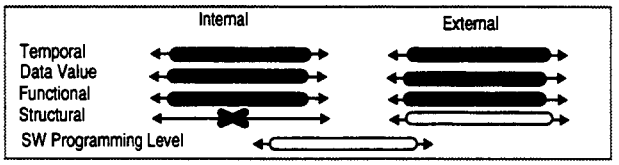
group gives a range of typical and extreme cases and identifies them wherever possible.

The group attempted to graphically depict the definition of each term relative to the taxonomy axes described in Section 2. The key in figure 2 should be used to interpret the graphic. Although some terms may span a range of abstraction levels, a given model instance describes information at one specific level within the span.

Figure 2 - Symbol Key:

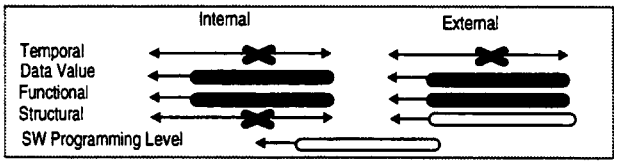
- Model resolves information at specific level relative to Table 1.
- Model resolves information at any of the levels spanned, case dependent.
- Model optionally resolves information at levels spanned.
- Model resolves partial information at levels spanned, such as control but not data values or functionality.
- Model does not contain information on attribute.

### 3.1 Behavioral Model -



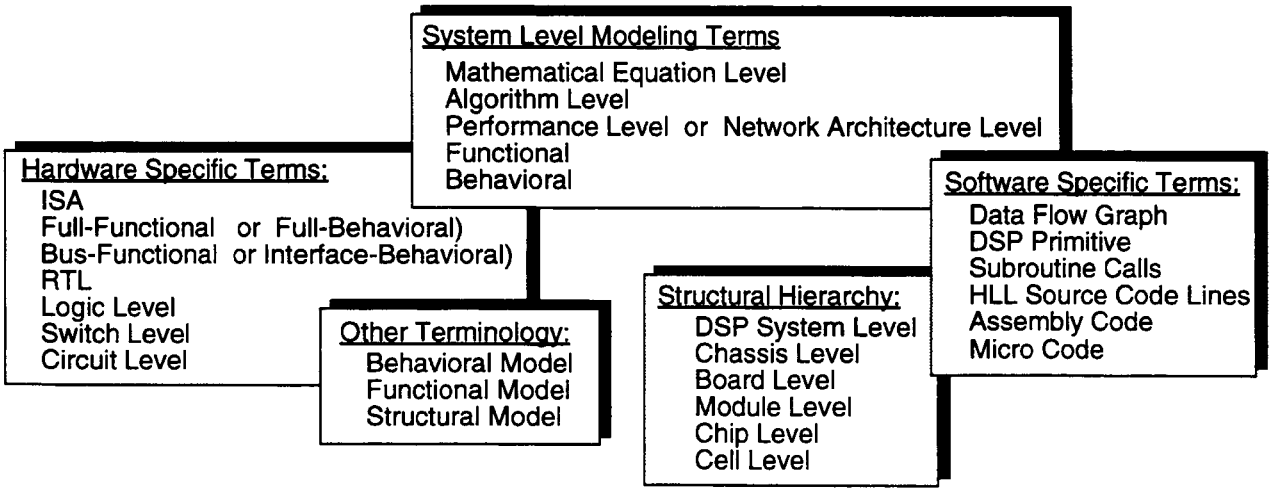

A behavioral model describes the function and timing of a component without describing a specific implementation. A behavioral model can exist at any level of abstraction. Abstraction depends on the resolution of implementation details. For example, a behavior model can be a model that describes the bulk time and functionality of a processor that executes an abstract algorithm, or it can be a model of the processor at the less abstract instruction-set level. The resolution of internal and external data values depends on the model's abstraction level.

### 3.2 Functional Model -

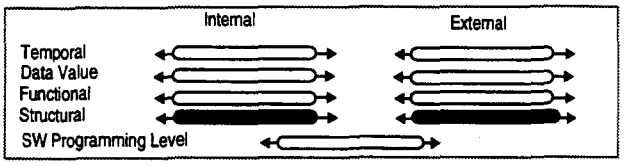



A functional model describes the function of a component without describing a specific implementation. A functional model can exist at any level of abstraction. Abstraction depends on the resolution of implementation details. For example, a functional model can be a model that abstractly describes the function of a DSP algorithm, or it can be a less abstract model that describes the function of an ALU for accomplishing the algorithm. The resolution of internal and external data values depends on the model's abstraction level.

**Table 2 - Commonly Used Modeling Terms**

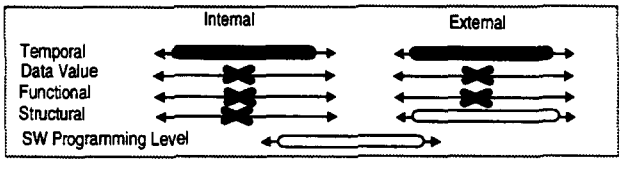


### 3.3 Structural Model -



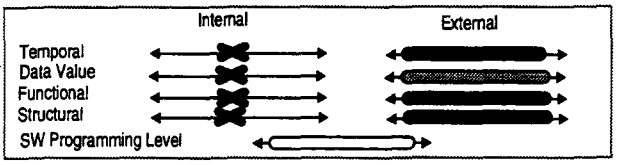

A structural model represents a component or system in terms of the interconnections of a set of components. A structural model follows the physical hierarchy of the system. The hierarchy reflects the physical organization of a specific implementation. A structural model describes the physical structure of a specific implementation by specifying the components and their topological interconnections. These components can be described structurally, functionally, or behaviorally. Simulation of a structural model requires all the models in the lowest (leaf) branches of the hierarchy to be behavioral or functional models. Therefore, the effective temporal, data value, and functional resolutions depend on the leaf models. A structural model can exist at any level of abstraction. Structural resolution depends on the granularity of the structural blocks.

### 3.4 Performance Model -



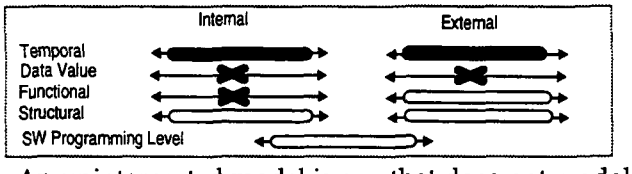

Performance is a collection of the measures of quality of a design that relate to the timeliness of the system in reacting to stimuli. Measures associated with performance include response time, throughput, and utilization. A performance model may be written at any level of abstraction. A highly abstract performance model can be a model that only resolves the time for a multiprocessor cluster required to perform major system functions, or it can be a less abstract model that describes the time required to perform simple tasks such as memory access of a single CPU. In the context of RASSP, however, the typical abstraction level of a performance model is often at the multiprocessor network level, also called a network architecture performance model. Internal and external data values are not modeled, except for control information.

### 3.5 Interface Model -



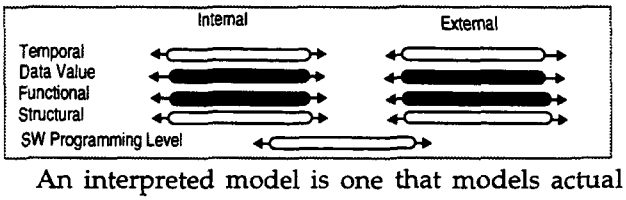

An interface model is a component model that describes the operation of a component with respect to its surrounding environment. The external port-structure, functional, and timing details of the interface are provided to show how the component exchanges information with its environment. An interface model contains no details about the internal structure, function, data values, or timing other than that necessary to accurately model the external interface behavior. External data values are usually not modeled unless they represent control information. An interface model may describe a component's interface details at any level of abstraction. The terms *bus functional*, and *interface behavioral* have also been used to refer to an interface model and are considered synonyms. The more general *interface model* name is preferred to the anachronistic *bus functional* term.

### 3.6 Uninterpreted Model -



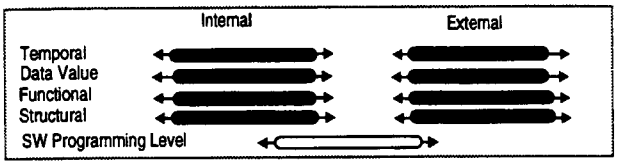

An uninterpreted model is one that does not model actual data values or data-related functionality either internally or externally. Only control information and control functionality are modeled.

### 3.7 Interpreted Model -



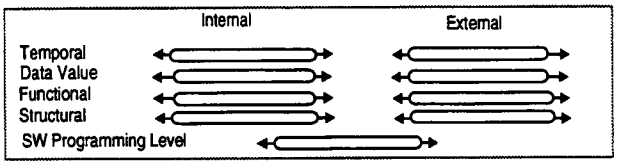

An interpreted model is one that models actual data values and data-related functionality and control-related functionality both internally and externally.

### 3.8 Virtual Prototype -



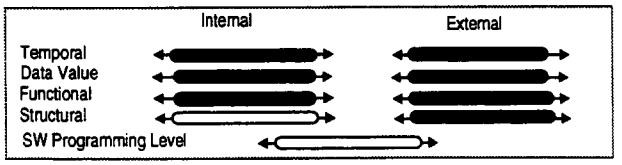

A virtual prototype is a comprehensive model of a system or component which models the external and internal temporal, data-value, functional, and structural combination of aspects for the system or component. A virtual prototype may be written at any level of abstraction. The virtual prototype is distinguished from the other types of simulation models, which describe only singular or a few aspects such as timing, function, or structure, in that it combines multiple aspects into a single simulation model. In terms of the other model types, the virtual prototype may be defined as an interpreted behavioral structural model of a system. The most significant usage of the term *virtual prototype* occurs at and below the network architecture level.

### 3.9 Mixed-Paradigm Model -



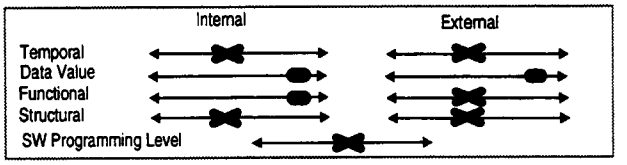

A mixed-paradigm model is a combination of models of differing paradigms. Such a model has formerly been called a *hybrid* model. However, the term *mixed-paradigm* is more precise. The term *hybrid* has also been used to describe a mixed-level model containing constituent component models of differing abstraction levels. Because the term *mixed-level* model is a less ambiguous direct synonym for such a model and has been in use for many years, it is preferred over *hybrid* in describing such models to minimize the proliferation of identical terms.

### 3.10 Full-Behavioral Model -




The full-behavioral model is a component model that exhibits all the documented timing and functionality of the modeled component, without specifying internal implementation details. This type of model has traditionally been called a *full-functional* model and is therefore a synonym. However, the newer term is preferred for its greater accuracy and consistency to the definitions of its constituent terms.

### 3.11 Mathematical-Equation Model -

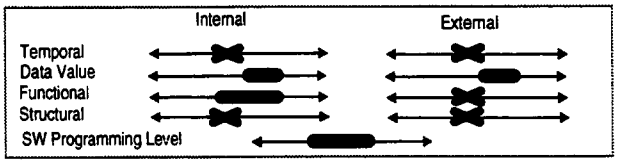



The mathematical-equation level of abstraction describes the functional relationship of data values. A mathematical-equation level description is a purely algebraic expression of the function the target system is to solve. The mathematical model is differentiated from an algorithm description in that a mathematical model does not imply a specific sequence of operations to implement the function. Examples of mathematical descriptions for system functions are:

$$y = sqrt(x) \quad or \quad y = sin(x)$$

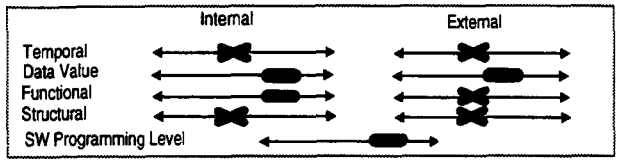
These functions represent well-defined mathematical relationships but do not indicate methods for their computation, for which their are many, such as lookup table, Newton's method, or Taylor-series expansion.

### 3.12 Algorithm Model -



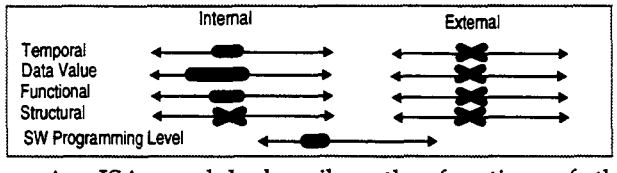

The algorithm level of abstraction describes a procedure for implementing a function as a specific sequence of arithmetic operations, conditions, and loops. An algorithmic description is less abstract than a purely mathematical description because it provides more detailed information for implementing the function(s). An algorithm model transforms actual data. Examples of algorithms are bubble-sort, Givens triangularization, Cholesky matrix decomposition, bisection method, Cooley-Tukey FFT, and Winograd FFT [8].

### 3.13 Data Flow Graph (DFG) Model -



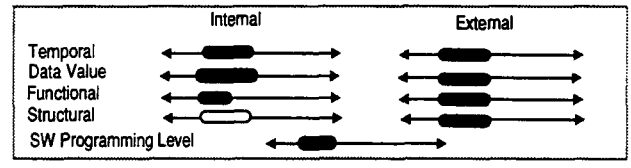

A DFG model describes an application algorithm in terms of the inherent data dependencies of its mathematical operations. The DFG is a directed graph containing nodes that represent mathematical transformations and arcs that span between nodes and represent their data dependencies and queues. It conveys the potential concurrencies within an algorithm, which facilitates parallelization and mapping to arbitrary architectures. The DFG is an architecture independent description of the algorithm. It does not presume or preclude potential concurrency or parallelization strategies. The DFG can be a formal notation that supports analytical methods for decomposition, aggregation, analysis, and transformation. The DFG nodes usually correspond to DSP primitives such as FFT, vector multiply, convolve, or correlate. The DFG graph can be executed by itself in a data-value-true mode without being mapped to a specific architecture, though it can not resolve temporal details without co-simulation with an architecture model.

### 3.14 Instruction Set Architecture (ISA) Model -



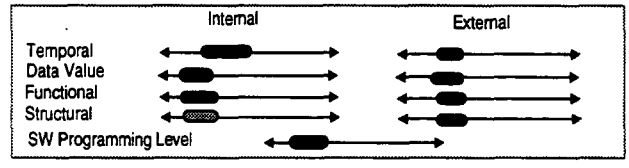

An ISA model describes the function of the complete instruction set recognized by a given programmable processor, along with (and as operating on) the processor's externally known register set and memory/input-output (I/O) space. An ISA model of a processor will execute any machine program for that processor and give the same results as the physical machine, as long as the initial states (and simulated I/O) are the same on the ISA model simulation as they are on the real processor. Such a processor model with no external ports is classed as an ISA model. If the processor model has external I/O ports, then it would be classified as a behavioral model. Data transformations of ISA models are bit-true, in terms of word length and bit values as observable in the internal registers and memory states. Port buffer registers, if modeled, are also bit-true. The temporal resolution of an ISA model is at the instruction cycle. Instruction cycles may span multiple clock cycles. An ISA model contains no internal structural implementation information.

### 3.15 Register Transfer Level (RTL) Model -



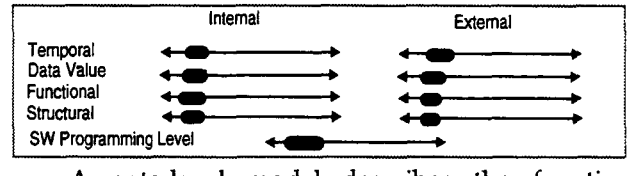

An RTL model describes a system in terms of registers, combinational circuitry, low-level buses, and control circuits, usually implemented as finite state machines. Some internal structural implementation information is implied by the register transformations, but this information is not explicitly described.

### 3.16 Logic-Level Model -




A logic-level model describes a component in terms of equivalent Boolean logic functions and simple memory devices such as flip-flops. The logic-level model does not describe the exact implementation in logic gates. The logic expressions can be transformed or reduced into functionally equivalent forms prior to target implementation in logic blocks.

### 3.17 Gate-Level Model -




A gate-level model describes the function, timing, and structure of a component in terms of the structural interconnection of Boolean logic blocks. The Boolean logic behavior blocks implement simple boolean functions such as NAND, NOR, NOT, AND, OR, and XOR. A gate level model describes the actual structure and versions of logic gates that are assembled to implementing the target component.

## 4 Conclusion

The scope of this article did not permit the complete listing of all the terms and definitions that the TWG has defined so far. However, a more complete listing of the current definitions is available on the following World-Wide-Web page: http://rassp.scra.org. Definitions for many supporting terms that are also very important throughout RASSP and yet are often problematic, can be found on the web page as well. These terms include:

system, architecture, component, module,
data, control, hardware, software, firmware,
test bench, validation, verification,
electronic data package
interoperability, top-down design, codesign.

The TWG task will continue throughout the duration of the RASSP project, so the definition document is a *living* document. The terminology has already gone through several revisions. However, note that the definitions above are subject to change as better definitions evolve.

During the next year, the TWG will continue to refine the current terms and add new ones as required by RASSP. The TWG will also consider other additional model attributes to the model taxonomy relating to a model's: maturity or validation level, simulation efficiency portability, complexity/size/lines-of-code, maintainability, flexibility, modifiability, expandability, and licensing cost.

Although these attributes appear to be useful in selecting, building, or using models for appropriate purposes within the design process, some are less easily quantified than others. For instance, simulation efficiency could be specified in events, cycles, or instructions per simulation-host CPU cycle, with memory requirements for the model's program code and data. Comparably objective units for portability or maintainability are less obvious.

In the near future, the TWG will use the refined nomenclature to define how the various types of models are implemented in VHDL and how interoperability between models of the same and differing types is obtained, as well as to identify what design risks and benefits they address.

Continued feedback and comments from the RASSP community are strongly encouraged to help reach a solid consensus on a collective terminology. Please direct your comments to the TWG in care of Carl Hein, at chein@atl.ge.com.


### Acknowledgments

The TWG gratefully acknowledges the contributions of Randy Harr, Arne Bard, John Hines, J.P. Letellier, Anthony Gadient, Maya Rubeiz, Gerry Caracciolo, and the many others who have provided valuable assistance in defining the taxonomy and terms and supporting the effort.


### References


[1] IEEE Std EIA 567 (August 1994) and IEEE 100
[2] Preliminary VHDL Modeling Guidelines, European Space Agency / ESTEC
[3] Army Handbook - The Documentation of Digital Electronic Systems With VHDL
[4] TIREP (Technology Independent Representation of Electronics Products) NSWC
[5] Madisetti, V., " System-Level Synthesis and Simulation VHDL: A Taxonomy and Proposal Towards Standardization", VIUF Spring, 1995 Proceedings.
[6] "Modeling and Simulation", Texas Instruments Semiconductor Group, 1990.
[7] Armstrong, J., "High Level Generation of VHDL Testbenches", Spring 1995 VIUF Proceedings.
[8] Ecker, W., Hofmeister, M., "The Design Cube - A Model for VHDL Designflow Representation", Proceedings of the EURO-VHDL, 1992, pp. 752-757.
[9] Famorzadeh, S., et.al., "Rapid Prototyping of Digital Systems with COTS/ASIC Components", Proceedings of RASSP Annual Conference, August, 1994.
[10] Blahut, R., Fast Algorithms for Digital Signal Processing, Addison Wesley, New York, 1985.