System-Level Synthesis and Simulation in VHDL - A Taxonomy and Proposal Towards Standardization

Vijay K. Madisetti
Electrical and Computer Engineering
Georgia Tech, Atlanta, GA 30332-0250
Proceedings of VHDL International Users' Forum, Spring 1995

Abstract—As part of US Department of Defense's RASSP program and also within the industry, VHDL is finding increasing use in the co-specification, co-design, co-simulation, and verification of embedded hardware/software systems consisting of multiple components (processors, memories, etc.), ASICs, and diverse interconnection and backplane options. Typically, VHDL models of hardware components are used to debug software design and ease integration and test, in what is essentially a hardware-less co-design process called virtual prototyping. Models developed for virtual prototyping provide more information about the components (and their internal state) being modeled than that required by EIA 567. The electronic data sheet of EIA 567 provides pin-based timing, physical, and electrical packages for components, while models used in virtual prototyping provide an augmented electronic data sheet that includes so-called virtual pins that probe internal state for purposes of system design and test (but are not included in the component's manufactured form). This paper proposes a systematic and concise draft standard for models used in system-level synthesis and simulation that augments EIA 567 to include virtual views within an augmented electronic data sheet. This draft proposal also establishes the levels of fidelity required for VHDL models for electronic design and simulation at multiple levels of abstraction. The model developer can also use this proposal to convey the level of fidelity incorporated into a model facilitating documentation and interoperability.

I. INTRODUCTION

System-level co-design and synthesis includes the modeling of digital systems in execution at multiple levels of abstraction [1]. Models for components range from gate-level models for ASICs to performance models of large multiboard multiprocessors. This author does not wish to burden the erudite reader with any more introduction than is necessary, as an extensive list of references exists with respect to multilevel VHDL models. In addition, the list of organizations that are attempting to resolve the modeling issue range from the Army, Navy, Air Force, EIA, and ESA [4]-[7], to name just a few, underlining the importance of the problem. While initial guidelines, such as Department of Defense’s DOD 80811, were proposed more for contractual procurement purposes than for simulation, EIA 567 attempts to address simulation issues as well, and is augmented in this draft of a proposed VHDL modeling standard for RASSP models targetted for system-level prototyping.

This interest is a blessing and also a curse, since the terminology and the framework of modeling in these references are not uniform; often several designers either refer to the same model by a number of different names, or imply different models by the same name. Either of these problems is serious enough, and when compounded with the issues of library generation, maintenance, and interoperability within a virtual prototyping environment, the resulting chaos can impede the rapid progress in methodology that RASSP is attempting to achieve.

After requesting the indulgence of the kind reader to refrain from casting away this paper as “yet another attempt to clear the muddy waters of VHDL modeling”, this author attempts to resolve this issue using the following approach:

1. A new draft standard for VHDL modeling in RASSP is proposed that augments an existing standard EIA 567 (August 1994). This augmentation extends the scope of the ELECTRONIC DATA SHEET.
2. A systematic basis for describing the fidelity and scope of a VHDL model within the proposed standard is established,
3. We redefine common VHDL models with this basis, more to establish the validity of the basis than from a desire to foist a new terminology on the reader. This new taxonomy also establishes a formal condition for the interoperability of RASSP models.
Indeed, the reader is encouraged to formulate his definition of any model in this new basis in an attempt to clearly communicate to others his model's scope and capabilities. Another reader may want to use this classification to explore a range of modeling capabilities systematically to ensure compliance with her application.

II. VIRTUAL PROTOTYPING IN RASSP

Rapid Prototyping of Application Specific Signal Processors (RASSP) (URL: rassp.scca.org) is a U.S. Department of Defense program aimed at dramatically improving the methods by which embedded systems are designed, maintained, and upgraded. In a candidate RASSP design flow of Figure 1, the primary difference from current systems design practice is that the hardware fabrication and assembly at the subsystem and system-level is eliminated from the in-cycle design loop [11]. The software is executed (and designed) on virtual hardware (in the form of VHDL models or hardware models and emulators) long before any HW fabrication and assembly is begun. This "virtual prototyping" environment significantly speeds up the HW/SW co-design and co-verification cycle through the use of models at multiple levels of design abstraction in the constituent VHDL libraries. The board-level and the multi-board integration is simulated and tested, additional control and diagnostic software is developed and debugged entirely in a user-friendly software environment. If the model libraries were accurate, the next stage could itself be that of the field prototype. A number of advantages can result from the RASSP design flow that are documented in [1].

In addition to virtual prototyping, Figure 1 introduces an additional stage called conceptual prototyping which involves early design, replacing the manual HW/SW partitioning commonly used in current practice. Conceptual prototyping utilizes automated tools that allow rapid estimation and evaluation of algorithmic, functional, architectural and enterprise-related tradeoffs early in the design process. A few candidate conceptual prototypes are then culled from the dozen or so generated at this stage, and then passed on to the virtual prototyping stage. Here, extensive evaluation and detailed design is done in virtual hardware and software leading to successful and rapid integration, again through the use of HW/SW reuse libraries, interoperable tools,

![Diagram of RASSP design process](image)

**Fig 1.** The author's vision of a mature RASSP Design Process (Fall 1997-98) with hardware-less in-cycle HW/SW co-design loops, enterprise integration, interoperable tool suites, automated metrics collection, and an additional stage for rapid early algorithm, functional, architectural, and automated HW/SW partitioning — "conceptual prototyping".

A Limitations of EIA 567

Concisely put, EIA 567 models the timing, electrical and physical characteristics of each physical pin provided by the manufacturer. While this may be sufficient for procurement and interface-level simulation purposes, it is insufficient for RASSP. In systems synthesis, the model must provide valuable information on the internal state and timing of the component (including that which is not accessible through the provided pin list). This additional information is valuable for system debugging, system synthesis, timing synthesis, and system verification within a virtual prototyping environment. As an example, the entity of a RASSP component model (FFM) for the Intel i860XP RISC processor developed at Georgia Tech's DSP Laboratory is shown below. In addition to the PORT description that corresponds to EIA 567, additional signals (although modeled as internal signals, they can be easily included in the POUI declarations) that model internal state (busses, registers, etc.) are also provided to allow the system prototyper obtain information that would be useful for algorithm, and system-level design and verification. For instance, using the VIRTUAL VIEW, the systems designer can obtain the state
use work.I860MATHPACK.all;

entity i860 is
generic (CLK_PERIOD : TIME := 25 ns; MEMORY_SIZE : INTEGER := 4096; START_INST_FILE : MEM_FILENAME := "MEM262143"; START_DATA_FILE : MEM_FILENAME := "MEM000000"; DEFAULT_TIME : TIME := 1 ns);

--- ******** PHYSICAL PINS OF I860XP ----

port (-- Clock and reset lines
    CLK   : in STD_LOGIC;
    RESET : in STD_LOGIC;
    -- Cycle control
    ADS_N  : out STD_LOGIC;
    BRDY_N : in STD_LOGIC;
    CACHE_N: out STD_LOGIC;
    LEN    : out STD_LOGIC;
    LOCK_N : out STD_LOGIC;
    NA_N   : in STD_LOGIC;
    NENEN_N: out STD_LOGIC;
    -- Address, data and byte enable lines
    ADDRESS: inout EX_ADDR_TYPE;
    BE_N   : out EXT_BYTE_ENA_TYPE;
    DATA   : inout EXT_DATA_TYPE;
    -- Cache control
    KEN_N  : in STD_LOGIC;
    -- Cycle definition
    MIO_N  : out STD_LOGIC;
    DC_N   : out STD_LOGIC;
    WR_N   : out STD_LOGIC;
    PCYC   : out STD_LOGIC;
    CTPY   : out STD_LOGIC;
    -- Interrupt signals
    BERR   : in STD_LOGIC;
    INT_CS8 : in STD_LOGIC);
end i860;

-- ******** PHYSICAL PINS OF I860XP --- architecture BEHAVIOR of i860 is

-- *** VIRTUAL PINS - BEGIN ********

--signal FL_REG : FL_REG_ARRAY;
--signal IN_REG : IN_REG_ARRAY;
signal EXTERNAL_BUS_OP : BIT_RESOLVE;
signal EXTERNAL_BUS_OP2 : BIT_RESOLVE;
signal EXTERNAL_CODE_READ : BIT_RESOLVE;
signal DONE_SIGNAL : BIT_RESOLVE;
signal LOAD_STORE_START : BIT;
signal DATA_ADDR_SIG : INTEGER,
signal BYTE_NUM_SIG : INTEGER;
signal OP_SIZE_SIG : INTEGER;
signal TRANSACTION_TYPE : INTEGER;
signal PM_SIG : INTEGER;
signal BUS_DATA_SIG_2INT : INT_2D;
signal BUS_DATA_SIG_4INT : INT_4D;
signal PFLD_STAGE1_RES : INT_4D_RESOLVE := (others => 0);
signal MMU_START : BIT_RESOLVE;
signal PC : INTEGER := 0;
signal ADDRESS_INT_SIG : INTEGER;
signal ATE_SIG : BIT;
signal DUAL_MODE_SIG : BOOLEAN;
signal DATA_LOAD_IN_PROGRESS : BIT_RESOLVE;
signal DATA_STORE_IN_PROGRESS : BIT_RESOLVE;
signal INST_LOAD_IN_PROGRESS : BIT_RESOLVE;

-- History:
-- 3AUG93 added 3 VIRTUAL
-- signals to mirror
-- pc and current instructions
-- to use set PC_OFFSET to location
-- of first line of
-- compiled code. this will allow
-- ease of reading when
-- comparing to assembler listing

signal PC_OFFSET_SIGNAL : INTEGER := 0;
signal PC_TIMES4_SIGNAL : INTEGER;
signal CURRENT_INTR SIGNAL : BIT_32;

-- Initialize to a reserved instruction
-- format 0 nothing is executed
-- when this is done. This is the core
-- escape instruction with bits
-- 4 down into 0 equal to "00000"

signal FETCH_BUFF : INT_2D := (others => DUMMY_INST);
signal RELOAD_FETCH_BUFF:
  RPLL_2D := (others => FALSE);
signal FETCH_INSTR_EXT : BIT := '0';
signal FDEST_SIG : INTEGER;
signal INSTR_BUFF : INT_8D := (others => DUMMY_INST);
signal CACHE_FILL : BIT;
signal SAVE : BOOLEAN;
signal FILL_REG_FILE_FROM_MEM : BIT_RESOLVE;
signal DATA_LOAD_BUFF : INT_2D;
signal PHYS_ADDR : INTEGER;

-- ** VIRTUAL SIGNALS - END *****

begin -- behavior of i860
  -- Drive the data signal to
  -- zero vector everytime
  -- The resolution
  -- function will handle it.
  -- All zeros imply no effect.
  DATA <= (OTHERS => 'Z');
  -- "0000000000000000"

EXT_CODE_READ:
  process
    variable DONE : BOOLEAN;
    variable BURST_FILL : BOOLEAN := FALSE;
    variable BRDY_COUNTER : INTEGER := 0;
    variable i : INTEGER := 0;
    begin
      EXTERNAL_CODE_READ <= '0';
      wait until EXTERNAL_CODE_READ'EVENT and EXTERNAL_CODE_READ = '1';
      wait until FALLING_EDGE(CLK);
      EXTERNAL_CODE_READ <= '1';
      -- Actually BRDY_N should be
      -- low on the next rising edge of the
      -- clock in order to correctly load
      -- the data but we will assume
      -- this is the case.
      -- This is done because
      -- on the next rising clock
      -- edge we want to fetch
      -- the next instruction.
      DONE := FALSE;
      BRDY_COUNTER := 0;
      BURST_FILL := FALSE;
      i := 0;
      while (DONE = FALSE) loop
        -- [ code deleted ]

II. AUGMENTING EIA 567

EIA 567 (August 1994) defines concepts, terminology and information required for constructing a VHDL component model to be in used in hierarchical design and simulated interoperably with other models conforming to this standard.

The basis premise of EIA 567 is that to simulate a complex system consisting of multiple components, we need to define common modeling interfaces, conventions, and simulation modes. These common bases ensure the new components can be
simulated together (primarily hardware models). Reuse is also possible. The goals of the proposed modeling standard are —

- Conform to common signal interface convention, (this interface need not be a bus),
- possess common simulation capabilities,
- are reusable as library elements of other designs,
- support multiple procurement sources,
- support technology independence,
- support VHDL modeling and simulation in system level synthesis and simulation, e.g., virtual prototyping.

These traits are of importance to any design philosophy, be it RASSP or otherwise. We now examine the use of the proposed augmentation to EIA 567 in defining commonly used models. The resulting augmented electronic data sheet is described (See Figures 2, 3, and 4).1

A. Glossary

1. COMPONENT — A component is any part of a physical design hierarchy that may be included one or more times in the design and specified independently for each appearance through use of a VHDL CONFIGURATION

2. LEAF LEVEL COMPONENT In a hardware design hierarchy this is a COMPONENT that is re-used without modification from a previous design, or a component that is irreducible part of a design. Examples - macrocells, etc.

3. PIN — A pin is an electrical connection to a physical component. Pins can be POWER pins, SIGNAL (DATA or CONTROL), ARTIFICIAL PINS, VIRTUAL, or NO CONNECT PINS.

4. POWER PIN — A power pin is an electrical connection through which electrical power is supplied to the device. POWER pins are not necessary for simulation.

5. SIGNAL PIN- DATA — This is an electrical connection through which the physical device exchanges application-related data information with other physical devices. Signal data pins are necessary for procurement and simulation. (These appears as ports in the entity description).

6. SIGNAL PIN- CONTROL — This is an electrical connection through which the phys-
ical device exchanges application and system-related control information with other physical devices. Since control information provided by one component may be interpreted as data by another component, both these pins fall under the general class of SIGNAL PINS. These pins are necessary for procurement and simulation.

7. **NO CONNECT PIN** — This pin is a designated pin in the physical VIEW of the device that is not connected to power and is not a signal (control or data) pin.

8. **TEST** or **ARTIFICIAL PIN** — This pin corresponds to the pin in the physical VIEW of the device that is not a SIGNAL PIN or a POWER PIN, and can be used for testing purposes or probing purposes. No connection to other devices is implied. All these pins shall be included on the component entity SIGNAL_PIN port declaration and have the attribute ARTIFICIAL.

9. **VIRTUAL PIN** — Those pins that are used for purposes of virtual prototyping, and do not have any correspondence to the timing, physical, and electrical views of the component model, but instead are introduced by the system designer (for purpose of simulation and verification). E.g., Performance Models use VIRTUAL PINS to communicate tokens. These pins can also be called PROBE PINS.

10. **OPERATING POINT** — An OPERATING POINT of a component is in a specific simulation condition.

11. **VIEW** — The VIEW is a set of logically related data that represents the significant characteristics of a component in operation.

12. **ELECTRONIC DATA SHEET (EDS)**

   The ELECTRONIC DATA SHEET is a set of VHDL packages that define a TIMING VIEW, an ELECTRICAL VIEW, and a PHYSICAL VIEW. An AUGMENTED EDS consists of an additional KERNEL VIEW, and an INTERFACE view of the component.

13. **ELECTRICAL VIEW** — An ELECTRICAL VIEW specifies the voltage and current characteristics for each SIGNAL (DATA and CONTROL) pin to the device.

14. **TIMING VIEW** — A TIMING VIEW specifies the signal propagation and timing constraints and behavior associated with each SIGNAL PIN (data and control) at a set of OPERATING POINTS.

15. **PHYSICAL VIEW** — Represents physical packaging characteristics of the device.

16. **KERNEL VIEW** — Represents the functional values (or state) and timing constraints and behavior of the each SIGNAL, POWER, TEST, VIRTUAL, PROBE, or ARTIFICIAL pin connected to the KERNEL (see Skillicorn’s classification in Figure 4) of the component.

17. **INTERFACE VIEW** — Represents the functional values (or state) and timing constraints of each SIGNAL, POWER, TEST, VIRTUAL, PROBE or ARTIFICIAL pin connected to the INTERFACE (see Skillicorn’s description) of the component.

18. **DESIGN UNITS** — consist of entity declarations, architecture bodies, configuration declarations, package declarations, and package bodies.

19. **DESIGN FILE** — A DESIGN FILE is a set of sequential text files that represent one or more VHDL DESIGN UNITS developed for a component as part of the RASSP modeling purpose.

20. **SOURCE LIBRARY** — A SOURCE LIBRARY is an implementation dependent storage facility for a set of DESIGN FILES containing one of more DESIGN UNITS.

21. **REFERENCE LIBRARY** — This is an implementation independent storage facility for a set of executable models for components that can be simulated in a VHDL environment.

22. **COMPONENT MODEL** — A component model is used to define both the KERNEL and the INTERFACE functions (and views) of a RASSP COMPONENT. Each COMPONENT MODEL shall consist of at least one entity-architecture pair, an ELECTRONIC DATA SHEET (AUGMENTED or otherwise), a TEST BENCH, and supporting DESIGN FILES.

23. **INTERFACE MODEL (IM)** — An INTERFACE MODEL is a MODEL used to define the operation of a COMPONENT with respect to its surrounding environment. The details (see section on fidelity) of the INTERFACE are provided to show how the component exchanges information with its environment. KERNEL VIEW may or may not be included, though INTERFACE VIEW is necessary.

24. **BUS FUNCTIONAL MODEL (BFM)** — The BFM is a IM where the surrounding
environment is a bus. VIRTUAL pins may stimulate the BFM.

25. FULLY FUNCTIONAL MODEL (FFM) — The FFM is a component model that contains all the documented complexity of the device. The details of the device are such that it mimics the documented complexity. Documented aspects of the KERNEL VIEW and the INTERFACE VIEW are provided in addition to the other views. This is essentially an AUGMENTED ELECTRONIC DATA SHEET.

26. PERFORMANCE MODEL (PM) — The PM is an IM in which only the INTERFACE VIEW is necessary. Other VIEWS are optional. ARTIFICIAL, TEST or PROBE (VIRTUAL) pins may stimulate a PERFORMANCE MODEL.

27. INSTRUCTION SET ARCH MODEL (ISA) — The ISA is a component model that contains part or all of the documented complexity of the device. See the next few section for more details.

28. TEST BENCH — A TEST BENCH is a collection of VHDL DESIGN UNITS and supporting data files that apply stimuli to the model under test, and compares observed response with expected response, and reports any differences.

29. SIGNAL CLASS — A SIGNAL CLASS is a group of signals that share characteristics (and attributes, as defined in the next section).

The reader may note that a number of definitions of models (especially, PM and IM, ISA and FFM), tend to overlap. This is an issue, not one of scope, but rather one of fidelity of each VIEW in the AUGMENTED ELECTRONIC DATA SHEET (Figure 2). This proposal also resolves the issue of overlap via the following “attribute”-based description.

III. ATTRIBUTES OF A RASSP MODEL

Any sequential digital component in execution can be represented by its internal state, and its inputs/outputs. The inputs and outputs can be control, data, or addresses, and the state represents information stored in available registers and memory elements of the component. A VHDL model (at any level of abstraction) attempts to capture the state and outputs/inputs with some degree of fidelity. The VHDL model has a level of fidelity that can be described by the following four attributes (Figure 1)².

1. Value — Most variables, data, addresses, procedures and functions (such as the FFT or DCT) have numerical values. The model can choose to represent these values according to their true value. Alternatively, the model may choose to represent only some (or none) of the signals and state variables correctly in terms of numerical value (i.e., equivalent to the values from an executable specification). For instance, some signals (inputs or outputs) could be modeled with numerical precision, while others can be either wrong or not modeled at all — leading to the designation: partially true. These two classes (not necessarily exclusive) are represented in Figure 5 as \(v_0\) and \(v_1\), respectively. The terms interpreted and uninterpreted have also been used for \(v_0\) and \(v_1\) in recent performance analysis literature.

2. Format — Signals (inputs and outputs) and the state variables can be modeled in a variety of formats. These could be bits (i.e., 0/1), bit-composite vectors (representing busses, arrays, etc.), or of an abstract type (integer, real or enumerated types). These are represented

²The experienced reader may propose, with this author's support, that the cardinality of the set of attributes need not be limited to the suggested four.
in Figure 5 as \( f_0, f_1, \) and \( f_2 \).

3. **Timing** — A model can represent timing with one or more levels of fidelity. For instance, a flip-flop can be modeled with timing that is clock-related, while a coarse-grained task in a parallel program can be modeled in wallclock seconds. Combinational circuits can be represented via gate-propagation delay. The user can define generics for timing (setup, hold, timeouts, etc.). Bus-based protocols may require the fidelity at the granularity of event-driven timing (as in asynchronous handshake protocols). Algorithmic descriptions require the specification of timing for causality. The latter ensures precedence of computation or control activities. For instance, if \( A = B + C \), then \( B \) and \( C \) are evaluated before \( A \) is, preserving causality. These levels are represented as \( t_0, t_1, t_2, t_3, t_4, t_5 \) in Figure 5. A single model may choose one or more levels of timing fidelity.

4. **State** — It may be useful to depict the typical digital system being modeled according to Skillcorn’s architectural description [10]. In this description, digital systems can be partitioned into a datapath (or data processor), instruction processor (or controller), data memory (for operands) and instruction memory (for instructions and control), and an interface that allows the digital system to interact with the external environment (See Figure 4). Both ASICs and complex COTS RISC processors and single and multi-board computers can be described using Skillcorn’s classification [10]. The state of each of these subcomponents of a digital system — Datapath (DP), Instruction Processor (IP), Data Memory (DM), Instruction Memory (IM), or the Interface (IF) — may be of interest to the RASSP system designer.

This state may be modeled partially (especially in the case of interface models), or at the level of programmer visibility (in COTS processor ISA models), and also at the level of the complete internal state (in gate-level and RTL models). Finally, the state may not be modeled at all. These four (possibly non-exclusive) levels are described as \( s_0, s_2, s_1, \) and \( s_0 \) in Figure 5.

When we refer to state, we have the option of further refining this attribute to associate it with each Skillcorn component of a digital system. For instance, a certain model may accurately provide the true value of the component’s interface, while only providing partial information about the state of the datapath or the controller. The latter information would, however, be useful when using a model to simulate the numerical performance (e.g., 16-bit fixed point) of a COTS part.

We, for the purposes of this taxonomy, simplify Skillcorn’s description by partitioning the digital system being modeled into two subcomponents - (1) the interface (IF), and (2) the kernel (KR). KR includes the datapath, controller, data memory, and instruction memory and associated internal logical structure, including those aspects that are visible to the programmer.

### IV. Examples

We will now describe some examples\(^3\) illustrating the use of Figure 5 in the precise description of a RASSP model of Figure 2. Using Figure 4 as a reference, each model defines the fidelity of the internal kernel (KR) and the interface (IF), separately.

Let us assume that we are modeling a COTS RISC processor at the level of the Instruction Set Architecture (ISA) [5]. A precise description of the ISA model using this new framework is given as follows\(^4\).

\[
\text{ISA - KR} = \left[ v_0, - - - - , (t_2, t_3, t_4), s_1 \right]
\]

\[
\text{ISA - IF} = \left[ v_0, - - - - , - - - - , s_1 \right]
\]

The ISA model, therefore, models the exact computation in terms of numerical value (as if the program were executed on the actual COTS RISC processor) to same bit-level numerical precision (i.e., \( v_0 \)). The format of the input/output signals and the state can be represented with a wide degree of freedom as shown by “- - - -”, since all formats are possible within the model. The user may wish to model internal memory state as integers to reduce storage overhead, and data as bit-composite vectors, etc. The ISA model has the choice of one or more of \( t_2, t_3, \) and \( t_4 \). Thus the ISA model does not preserve clock-related or propagation delay based features of the COTS RISC processor being modeled, but instead uses wall-clock time, user-defined generics or event-driven

---

\(^3\)The definitions of the terms used in this section are consistent with proposed EIA 567-A and other guidelines [3]-[7]

\(^4\)(\( x, y, z \)) implies one or more of \( x, y, \) or \( z \).
(interrupts, traps, resets, etc.) timing constructs to represent the timing behavior of the model. Finally, the ISA models the programmer-visible states of the COTS RISC processor (during execution) only, and not the complete Kernel state, as described in the above description. The value of the Interface (IF) state (e.g., pins) can also be modeled partially, and the data and address pins may have true values, though not related to a clock edge. Some of the pins of a chip may not be modeled at all (e.g., Read/Write, CS, DMA ports, etc.). We can thus very precisely define the scope and fidelity of an ISA model, with little room for doubt.

To further highlight the utility of this framework, let us examine the representation of an RTL model:

\[
RTL - KR \equiv [v_0, \ldots, (t_1, t_2, t_3, t_4), s_0]
\]

\[
RTL - IF \equiv [v_0, \ldots, (t_1, t_2, t_3, t_4), s_0]
\]

The RTL model, therefore, includes the clock-related behavior of the COTS RISC processor while executing application code, in addition to preserving the entire internal state (including internal registers and memory contents, that are not normally visible to the programmer). Both the RTL and the ISA model preserve the correctness of the computation [5]. The RTL model, in addition, could be synthesized using high-level synthesis (HLS) tools.

The Bus Interface Model (BIM) of the same COTS RISC processor can be described by Figure 5 as [5,6]:

\[
BIM - KR \equiv [v_1, \ldots, \ldots, s_7]
\]

\[
BIM - IF \equiv [v_1, \ldots, \ldots, (t_1, t_2, t_3, t_4), s_1]
\]

This description implies that internal (i.e., KR) numerical values need not be correctly preserved within a BIM. However, timing at the interface is preserved with a great degree of fidelity, with a wide range of clock-related to event-driven (as in asynchronous handshakes for bus protocols. The Bus Functional Model (BFM) is identical to the BIM [7].

The Fully Functional Model (FFM) of a COTS RISC processor can be defined as [4]:

\[
FFM - KR \equiv [v_0, \ldots, \ldots, (t_1, t_2, t_3, t_4), s_1]
\]

\[
FFM - IF \equiv [v_0, \ldots, \ldots, (t_1, t_2, t_3, t_4), s_0]
\]

Thus the FFM is in some sense the union of the features of the ISA and the BIM of the COTS processor. Figure 6 shows the relationship between the FFM, BIM and the ISA models. There is a close relationship between the FFM and the RTL level model, except that the latter models the complete internal state.

The Performance Model (PM) can be described (as the reader will have realized):

\[
PM - KR \equiv [\ldots, \ldots, \ldots, \ldots, \ldots]
\]

\[
PM - IF \equiv [v_1, \ldots, \ldots, (t_2, t_3, t_4), s_2]
\]

Consequently, the performance model is used in early design of the RASSP prototype architecture, where rough estimates of utilization, throughput, and latency can be computed on the basis of wall-clock times, etc., with incomplete numerical and internal state precision [3]. If the user wishes to specify a varying level of internal and algorithmic precision in addition to a higher fidelity in timing, all that has to be conveyed are these additional attributes as part of PM-KR. This would not have been feasible with an imprecise “catch all” description of a performance model (PM) as found in recent literature. This capability to accommodate designer-specified fidelity is a strength of this proposed taxonomy. The designer thus is not restricted to implementing her model using a fuzzy terminology such as ISA or BFM, but through a precise attribute-based designation that is suited to her prototyping application, which could then be translated into English.

Finally, a high-level language (HLL)-based executable specification (e.g., a C program) can be represented in the proposed framework as:

\[
HLL - KR \equiv [v_0, f_2, (t_3), s_7]
\]

\[
HLL - IF \equiv [v_0, f_2, (t_3), s_2]
\]

We note that in Figure 5, as one moves from the lower half of the figure to the upper half, the level of detail with respect to structure is increased. Automated synthesis of systems would be feasible from this increased level of detail (e.g., as in the RTL model).

**Interoperability**

A necessary condition for the interoperability of two models is that their "timing" and "format" attribute sets have a non-empty intersection. Thus this proposed framework simplifies the determination of interoperability. The reader can easily build upon this basic definition, and we will establish the notions of interoperability in later papers. These interoperability definitions can be utilized in the automatic generation of multilevel VHDL models based on leaf-level models of varying abstraction.  

\[\text{Note that this is not a sufficient condition. The latter re-}\]
The modeling taxonomy proposed in this paper tells you the essentials of what the model is trying to capture about the real system (e.g., the roar of the tiger). The fact that the roar was stored on a Sony, tape-less recorder is not important (except to one who is developing another model, or a better one). The Ecker Cube, as we describe in [8], concentrates more on how the model was constructed, i.e., Sony and tape-less, (by specifying the way the model was constructed in behavior, dataflow or structual VHDL, rather than the fidelity with which it is modeling the real system). Once an “implementation independent” modeling nomenclature is adopted, the next step would be to use the Ecker Cube to describe its implementation in a (HDL). The latter step is analogous to selecting the right audio recorder, with appropriate performance level required for the application. Both steps are important. For instance, the ISA model can be implemented in behavioral view, while the FFM can be implemented by a dataflow view in VHDL, as defined in the Ecker cube. The reader may also refer to [9] for a description on how to implement behavioral and dataflow models for COTS components in VHDL.

In this proposal, the interaction between software and hardware is captured via attributes, value and state, that are not utilized in other models. Figure 5 also includes a “non-exclusive” descriptive space, wherein attributes such as \( v_0 \) and \( v_1 \) are not necessarily exclusive. This latter feature is useful in describing the iterative (and increasing) refinement of abstraction that is utilized in top-down design approaches, HW/SW codesign, and Co-simulation.

V. Example of Virtual Prototyping
The BIM models the timing behavior of a component’s interface (with respect) to its surroundings. Thus timing and format of output bus drivers for data, control/addresses, etc. are modeled as accurately\(^6\) as possible, while internal structure, state, or algorithmic values are not necessarily modeled. Thus BIMs are more suitable for providing rough estimates of system performance during

\(6\)While BIMs for SSI and MSI parts are often accurate in terms of timing, complex VLSI parts (i.e., RISC processors, DSP chips, or communication routers) whose timing is both data- and internal state-dependent (cache, interrupts, resource contention, etc) can seldom be accurately modeled by BIMs. Therefore, the author recommends the use of FFM's for complex VLSI parts. Hardware modelers provide limited internal (virtual) information, and their speeds (instructions/sec) are limited by network transfers and memory available.
ing conceptual prototyping. The FFM models **all documented** complexity of a physical component in a VHDL behavioral description. All timing and internal register states are modeled, in addition to full functional emulation of hardware behavior. In the virtual prototyping example of the this section, a FFM of the i860XP (developed at Georgia Tech), and BIMs of the memory, memory controller, FIR chip with buffer, and decoders were used. The board-level system was designed and debugged completely within a software-only environment using a VHDL simulator [2] within four man-days. Simulations speeds of 1000-2000 instructions/sec were typical.

### A. HW/SW Codesign of DSP board

A high-level description of the target system is shown in Figure 7. An equalizer is implemented as a FIR filter. Its design is to be finalized during the virtual prototyping effort. The FIR is loaded with incoming data and stores its output in a 128 word buffer. The i860 should read the data from the FIR’s buffer (by selecting its buffer for a read at an appropriate time). The buffer writes the data onto the data bus which loaded by the i860XP into its appropriate internal registers. After processing the data (the design must allow the capability to change this processing software easily at the user’s discretion), the i860 selects its local memory (RAM) (and de-selects the FIR’s buffer). The memory controller provides the appropriate control signals that allow the i860 to interface with a static RAM. The lower significant address bits are used to store the data in the RAM. The entire cycle then begins again, with the FIR processing new data in the meanwhile.

This example is typical of the functionality required of board level designs, and was chosen to be specific enough to highlight the different trade-offs that are available during the virtual prototyping process.

### B. RASSP Model Integration

All the VHDL models (FFM or BIM) were integrated to virtually prototype a DSP board with an-off-the-shelf RISC processor, memory, a custom memory controller, and a FIR filter ASIC with internal buffer. The operations of the virtual prototype (to be verified and evaluated) can be described as follows — the FIR filter receives data from an external source, filters the result and upon request will provide the data to the i860 in a first-in-first-out (FIFO) manner. The i860 processes the data and will store the result back to its local memory. The FIR filter is mapped off the memory space of the i860, and should be capable of directly connecting to the i860 with no extra glue hardware. In order to test the board, some software was written for the i860 as a compiled program that would perform a read operation to the address where the FIR filter was mapped, and then store the result in the local memory immediately. After the load and store operations, the i860 idles for 30 cycles, before starting over. The FIR and the i860 operate at 2.667 MHz and 40 MHz, respectively.

**Timing Verification**

Once the models were integrated together, the virtual prototype was simulated and debugged for correct operation.

**Proper Timing**

Two snapshots of the signal drivers on our virtual prototype is shown in Figure 8. The timing diagrams show the state of all related signal drivers at the time the i860 is trying to read data from the FIR. The i860 initiates a read by placing the FIR’s address on the bus and activating the address strobe signal, _ADS_N_. This can be seen in the timing diagram at time 11227.5 ns. The address lines are fed to a decoder that selects either the memory or the FIR filter by asserting either _FIR_CS_ or _MEM_CS_, respectively. After a

---

7 Nothing in the process of virtual prototyping precludes the use of any other COTS processor at any point during the design cycle, as long as its FFM is available.
delay of 5 ns (due to the propagation delay of the decoder), the FIR_CS signal is asserted by the decoder at time 11232.5 ns. After an additional delay of 5 ns (due to the propagation delay of the chip select logic in the FIR filter) at time 11237.5 ns the FIR filter begins driving the bus. Valid data is put on the bus by the FIR filter 5 ns after address strobe, ADS_N, is deactivated, at time 11257.5 ns. At the same time the BRDY_N signal of the i860 is pulled low by the FIR filter to notify the i860 that the data is ready on the bus. The i860 will then read the data on the next rising edge of its clock (or 11266 ns) and the read cycle is completed. The next operation after the read is a write to the memory, where the FIR data is put in the local memory of the i860 (Fig. 8, upper half).

Improper Timing

A common design error on any bus-based system is that of signal contention, when more than one source is driving the bus. In our virtual prototyping example, the hold time of the FIR filter was increased intentionally to create a bus conflict. Previously, the FIR chip would release the bus 5 ns after the chip was deselected (when FIR_CS was deasserted). In a new experiment, the hold time was increased to 40 ns. With identical application code executing on the i860XP, the processor will try to write the value (just read from the FIR filter) to the local memory. Concurrently (due to the increased hold time) the FIR filter is still driving the bus and a signal contention occurs. These points are also illustrated in the lower part of Figure 8 with the FIR_CS being deasserted at time 11266 ns. Adding a 40 ns hold time of the FIR will result in 11306 ns as the time at which the bus is released by the FIR filter. Concurrently, the i860 tries to perform a write to the memory at time 11277.5 ns as indicated by the ADS_N signal going low. The 16 bit data that is to be written to the memory is put on the upper data bits of the data bus (31 downto 16). At this time the lower 16 bits of the data bus are being driven by two different sources, i860 and FIR chip — one driving it with a value of 0x0000004600000000 and the other with 0x0000000000000004, resulting in the value of 0x000000460000000XX (Fig. 8, lower half).

VI. SUMMARY

A simplistic, and often loosely-defined, nomenclature of models used in computer science and engineering, (such as ISA, BFM, etc..) that has been enlisted to classify VHDL models used in system design, has been overused and misused in recent literature. The need for a flexible and precise standard for VHDL models for IIW/SW system design has been widely acknowledged within the user community. This paper has proposed a new standard that defines precisely the scope and fidelity of VHDL models for COTS components, (including non-combinatorial VLSI/MSI), utilized within the RASSP process. The internal kernel and the interface of the digital system being modeled are specified separately. The reader who is familiar with VHDL may quickly identify how the various attributes can be easily implemented within a simulation and synthesis environment [2]. The new taxonomic format for any model is determined by the fidelity of its AUGMENTED ELECTRONIC DATA SHEET (as per EIA 567):

\[
M - KR = [\text{VALUE}, \text{FORMAT}, \text{TIMING}, \text{STATE}] \\
M - IF = [\text{VALUE}, \text{FORMAT}, \text{TIMING}, \text{STATE}]
\]

It is hoped that RASSP participants may find this draft standard useful for concisely constructing their VHDL models in an unambiguous manner facilitating interoperability, consistency, and easy validation. It is expected that a number of other useful attributes will be proposed, together with an increasing refinement of the fidelity scale within each attribute. 8

VII. ACKNOWLEDGEMENT

The author is greatly indebted to valuable feedback from Dr. Randy Harr of ARPA-ESTO that contributed significantly to this paper (especially with respect to the partitioning of the interface and kernel within the VIRTUAL VIEW). The author is also indebted to Dr. John Hines (US Air Force) for his valuable comments, and for introducing the author to this interesting subject. Grateful thanks go to Mr. Thomas Egolf and Mr. Shahram Farnoush of Georgia Tech for many illuminating discussions on RASSP modeling. Comments (both brickbats and accolades) from Dr. Mark Richards of ARPA, Mr. Arne Bard of US Army Research Laboratory, Mr. Paul (Kassey) Kalutkiewicz of Lockheed Sanders Inc., Mr. Carl Heit of Martin Marietta Corporation, and Mr. Todd Carpenter of Honeywell, and Dr. Allan Anderson of Lincoln Labs,

8For the purposes of standardization, it is straightforward to convert the above model into an English textual description, where each of the various attributes is described and its scope and fidelity within the model is specified.

8.12
Fig 8. Example of Timing Synthesis in Board-Level Virtual Prototyping. Top half of Vantage Spreadsheet shows current timing, the bottom half shows improper timing.
improved upon the presentation and are acknowledged gratefully. The views of this paper are solely those of the author, and do not represent the views of the US Department of Defense, Lockheed Sanders Inc., or Martin Marietta Corporation.

References