1.0 Appendix D: Unresolved Issues

1.1 Regarding Naming

- The note on conditional expression designator indicates that the conditional expression will not be supported in all simulators. The same function can be done by using the conditional 'identifier'. Do we need to include this particular construct then?

1.2 Regarding Level 0

- Port Naming Conventions. Recommendations are OK, but there will be problems if these are enforced restrictions. The port names of ASIC cells are typically fixed earlier and independently of VHDL modeling considerations. To change port names in VHDL would introduce consistency problems.
- It's also not clear if VITAL places any restrictions on the _design_ (as opposed to models in the design). For example, must all signals, including those within non compliant models, be one of std_logic_1164's data types? Must all VITAL compliant model interconnections be without type conversion functions?
- Must a VITAL compliant system verify the stated compliance level, including issues like structure, naming conventions, and name length?
- The VITAL spec is defined using VHDL87, but we must ensure that VITAL models will run in a VHDL93 environment. Therefore, VITAL should also require that the models use no new VHDL93 keywords. Additionally, VITAL needs to consider how the spec will be extended to handle VHDL93, including extended identifiers and Pure/Impure functions.
- Section 2.2.4:It states "No subtypes are allowed" for std_logic_vectors. Why is there this restriction? I've seen many subtype declarations of std_logic_vector in order to defined a named constrained range, for example:

  ```
  subtype DBUS is std_logic_vector(31 downto 0);

  ...
  port(  ...
  D : inout DBUS;
  );
  ```

  There are many other uses/needs for subtype declarations, so why the restriction?
- It's not clear if VITAL compliant models can be intermixed with non-compliant models in the same design. We believe that intermixing should be allowed in order to support test benches, and design by incremental refinement.
- Timing Check Section: states that the "then" part of the "if" shall contain only calls to VITAL_Timing subprograms, yet this seems overly restrictive.
- Please describe how timing checks are applied to the netlist block.
- The signal naming convention is not specified, but doing so would greatly increase understandability. For example, the role of the *_ipd and *_zd signals used in the examples are not obvious to everyone.
- If "WIRE_DELAY" is the required name of the wire-delay block, that should be stated.

1.3 Regarding Level-1

- Cannot really see a need for the Level_1 attribute. Can't the analyzer simply revert to Level_0 if it doesn't pass all of the Level_1 requirements? And perhaps tell the user which level was recognized. This would be far cleaner from the model's perspective.
- Generics exist apart from back annotations. Back annotation influences the rules governing generic use, but the rules are still about generics. Please separate out the rules for generics from the rules for back annotations, and specify which rules apply to all generics, which rules apply to VITAL timing generics, and which rules apply only to generics which are back annotatable.
- It is not clear if both pin-to-pin (functionality / timing-checks and pin-to-pin delays) and distributed (netlist-block) sections can be present in the same Level 1 model. If both can be present simultaneously, it should be stated that these sections must be independent, that is, the pin-to-pin section and the distributed section pass no information between them and remain isolated from each other.
- In the VITAL specs, do we want to have "strictly enforcable" names for internal signals/variables like
  *_ipdfor delayed input signals
  *_zd for zero-delay output
  or should this be a "desired" convention ?
- Functionality section: Is there a need to limit the user to the kinds of statements that are mentioned ? Once we make the compliance guidelines for other sections fairly tight (using the points that I am outlining), the user should be allowed to use general behavioral code for his functionality. In fact, for other sections, it should be easy to build-in the compliance checks, but for the functionality section it may be difficult to enforce the rules that are mentioned now. Also, by making the functionality more general, in certain situations, he would be able to model things much more easily by using an internal signal (and making his process sensitive to it), which he would like to manipulate in the functionality block. Of course, he would benefit more if he uses stuff from the VITAL packages to describe his functionality.
- Section 3.6.3 In the current spec, the path-delay procedure will ALWAYS be activated. I would recommend that it should only be activated ONLY on outputs changes. As we have discussed in the past, calling it always may lead to inaccurate results. However, by calling it only when output changes, we can make the model inherantly more efficient, without losing out on any more accuracy.

1.4 Regarding VITAL_Timing

- Wanted a switch on the timing checks so that you can generate an X but NOT issue an assertion message. Disagreement with this suggestion, since the TimingCheck routines cause the X and that event should not go without a message indicating the violation. Don't just corrupt the result without giving a reason)
- The VitalExtendToFillDelay is most often a inefficent overhead. That is: Most cells do not generate 'Z' outputs and so only specify 2 delays rather 6. In the recommended style, all cells will call the ExtendToFillDelay function to "extend" the 2 delay values into an array a 6 delays. The additional 4 array entries won't be used. This operation of copying 2 values into 6 is done for each input on each output change. Possible Solution: We can easily take care of it by providing overloaded versions of PropagatePathDelay etc. so as to avoid the unnecessary call to ExtendToFillDelay. We would then have to overload CalcDelay, PropagateWireDelay as well
- We came to the conclusion that we can live with the non-support of 'PulseRejectLimit' (PRL) and 'Xlimit' for the time being. Right now, our ASIC libraries have requirements which are two extreme cases. (i) PulseRejectLimit = 0 ns, and Xlimit = 0 ns This is nothing but pulse-pass requirement,

and can be modeled using TRANSPORT delay construct in VHDL. (ii) PulseRejectLimit = <pathDelay>, and Xlimit = <pathDelay> This is nothing but pulse-suppress requirement, and can modeled using the 'inertial' delay construct in VHDL. For example, the statement Y <= (A AND B) after <pathDelay> will suppress pulses of width less than <pathDelay>. But (ii) is a workaround for the exact requirements, and ideally we should have support for PATHPULSE/GLOBALPATHPULSE constructs in SDF to meet these exact requirements. So, it is alright if PRL and Xlimit are not supported by VITAL for the time being. The current VITAL procedures can be used for the case (ii) without any problem. But to handle case (i), we need overloaded procedures which allow any pulse to pass to the output (TRANSPORT delays must be used in the procedures).

- While modeling the FIFO type IDT 7203 of Integrated Device Technology, Inc. on VITAL-specification, I found a problem with the "Generic Timing Parameter Form" of tpd. I couldn't define clearly some propagation delay timings. The customer defines on the reference: read falling to data the tpd Z to X with tRLZ and to Valid data with tA and read rising to data the tpd Valid data to X with tDV and to Z with tRHZ.
- The VITAL-specification presents the allowed parameter form(s) tpd, input and output definition but no reference to rising and falling of the input and the allowed parameter type(s) don't define the output changings Z, X and V (valid). The problems are multiple values on the same ports, different values on different reference edges and output changings of a busbetween Z, X and V. Is there a possibility to define other parameter forms and types?
- Pp 36: (d) What happens if the user does need -ve setup times ? Shouldn't we at least recommend a workaround ?
- Since the SDF doesn't support 'hi' 'lo' values for a given timing check should the procedures have only one value for the above ?
- I recommend, removing the VitalExtendToFillDelay routine, and instead overload the functions and procedures with parameters of type DelayType01Z or DelayArrayType01Z. These would need to be overloaded for DelayTypeXX, DelayType01, DelayType01Z, DelayArrayTypeXX, DelayArrayType01, DelayArrayType01Z. Unfortunately, this will require that these be declared a separate TYPEs rather than SUBTYPEs (this is why this is more than an simple change that might otherwise might have been feasable prior to Version 2.2). This has an apparent efficiency effect of not re-calling the VitalExtendToFillDelay function each and every time one of these routines (like VitalPropagateWireDelay) is called.
  Distributed delay models may need to include other than concurrent procedure calls to Vital primitives. Should the VITAL_NETLIST block also be allowed to include:
  a) Behavior process's,
  b) component instances ?

  As a result of the restriction that back annotation may be via a Configuration OR by direct read of the SDF, BUT NOT BOTH, what is the appropriate action for a direct read annotator (simulator) when values are also associated with generic timing parameter in a configuration. My recommendation is that the values from the configuration are ignored. This would imply that INCREMENTAL SDF values are added(subtracted) from the DEFAULT values of the generic timing parameters - not the actual (after eleboration)values.

1.5 Regarding Combinatorial Primitives within VITAL_Primitives

- Pullup and pulldown primitives could be added to the primitives package as these are used extensively in designs.

- Need for expanded description on primitive delay selection algorithm: 5.4.1 There is not really enough information in the spec to enable "selection of output time (and corresponding delay)" implementation. Again, do we have to make recourse to the packages provided? Also, the descriptions only deal with scheduling 0 or 1. What about scheduling an X?
- GlitchDetect: If implementers cannot change it, then why is it in the spec? How about including a parameter on the primitives to allow the selection?
- PrimGlitchMode: If implementers cannot change it, then why is it in the spec? How about including a parameter on the primitives to allow the selection?
- DFF & Latch: If we have the combinatorial primitives, why not include a DFF and Latch for completeness?
- Added Detail: We propose that for all the primitives the truth table should be provided so that there is no ambiguity in the behavior. It is not clear how they deal with X inputs, for example, without recourse to the packages.
- There are no RAM or ROM primitives provided, but RAM and ROM are important building blocks.
- Given that X inputs must be handled, it seems odd to support only 2 delay values (DelayArrayType01), when there will be more than 2 types of delay. See N-input Logical Gates

1.6 Regarding Tables within VITAL_Primitives

TruthTables

- VitalTruthTable: How often will the error regarding the size of the output vector be reported -- once per simulation, once per invocation?

State Tables

- If VitalStateTables are intended to represent Moore machines, then why do they allow matching on _state_ edges?
- VitalStateTables: Why is more than one input variable allowed to change at a time? These extensions add considerable complexity to the interface and the semantics, and seem unnecessary.
- Can we have a subtype VitalStateTable which just allows edge types? We don't really want to support all the symbols
- Transitions from/to 'X' logic state have yet to be added to the 'VITAL_Tables' package.
- The example shows the use of constants (instead of "literals"). In the error-trap section, we should mention that these constants should be locally static. In fact, it might even be simpler, if we restrict the table entries to be an enumeration literal only.

X propagation and Tables

Add a BOOLEAN parameter 'XReduce' to the VitalTruthTable and VitalStateTable sub-programs.

If the value of ‘XReduce' is FALSE, then the sub-program functions as currently specified. That is, find the first match and return the corresponding output value(s). If no match is found, return 'X' for all outputs.

If the value of 'XReduce' is TRUE, then the algorithm of the sub-program would be modified to reduce the pessimism in 'X' propagation as follows: a) If after the input data vector has been converted to X01, the value of an element is 'X', then two searches are performed: one with the ‘X‘ replaced by '0' and the other with the 'X' replaced by '1'. If there is an additional 'X' value in the resulting data vectors, each is similiarly converted into two separate searches. This is repeated for any further input 'X' values. [Note: 1 input 'X' results in two searches, 2 'X's results in 4 searches, 3 'X's results in 8 searches ... etc]

b) For each output element: the output values corresponding to each of the searches are compared. If all the values are the same, then that is the returned value for the element, otherwise 'X' is the returned value of the output element.

For example, take the following truth table for a 2:1 mux.

CONSTANT and_table : VitalTruthTableType(1 TO 4, 1 TO 4) :=

```
-- D0  D1   S   Q
( ( '0', '-', '0', '0' ),
  ( '1', '-', '0', '1' ),
  ( '-', '0', '1', '0' ),
  ( '-', '1', '1', '1' ) );
```

When D0 = '1', D1 = '1', S = 'X' :

a) If 'XReduce' is FALSE, then the input vector "11X" will not match any entry and so the returned value is 'X'.

b) If 'XReduce' is TRUE, then the input vector "11X" will translate into two searches (for "110", "111"). The first search ("110") will match the second table entry, the second search ("111") will match the fourth table entry. In both cases the output value (for Q) is '1', so the returned value is '1'.

Additional Notes:

1) The algorithm for handling input 'X' values in VitalStateTable may require some further consideration. My initial thoughts are that the replacement of 'X' by both '0' and '1' would be performed prior to translation to an edge condition. That is: if the previous value was '0' and the current value is 'X', this would result in a search for 0->1 and 0->0. A previous value of 'X' and a current value of '1' would result in searches for 0->1 and 1->1.

I believe this gives the X propagation I would expect, but I haven't done a complete analysis. For example, on a D-FlipFlop, if CLK='X', D=‘1', Q='1' then Q would remain at '1'.

2) The search (for X free inputs, or patterns created from X inputs) is the same : First match, based on the order of the entries in the table.

TruthTable entries with include an symbol which specifies a match to an 'X' value would never match when the XReduce = TRUE.

3) The 'XReduce' parameter MUST BE A CONSTANT and the associated actual value MUST BE LOCALLY STATIC.

Comments:

The algorithm for 'XReduce' = TRUE may appear to be a substantial runtime penalty (all that searching will run slow). This is possably true for the raw execution of the VHDL package code (although, the size of the corresponding truth table to achieve the same X propagation should also be considered). However, the key point is that the ‘XReduce' parameter allows the modeller to better communicate the "intent" of the model function. In this case the intent is the level of pessimism in propagation of X values. The modeller could implement the same functionality without the XReduce parameter, through a more explicit (longer) truth table. However, this requires analysis of the the truth table to capture the "intended" algorithm for X propagation.

Providing the EDA tool vendor a more abstract view of the intended model funtionality, should allow more oportunity for optimization / acceleration. Thus although the raw VHDL in the VITAL package body is more complex, the resulting cell model may be simplier / faster.

1.7 Regarding SDF to/from VHDL Mapping:

- I believe back-annotation should be described as a modification to the elaboration rules for the generic map aspect defined in the 1076 LRM.
- "... Elaboration of a generic association element consists of the elaboration of the formal part and the evaluation of the actual part. The generic constant or subelement thereof designated by the formal part is then initialized with the value resulting from the evaluation of the corresponding actual part." <Then add> ", unless there exists a back-annotation value for the generic in which case the generic constant is initialized with the back-annotation value."
- In this table, Default Value represents the default value specified when the generic was declared; GenericMap Value represents the value provided with a generic map (in either a component instantiation statement or a component configuration); and SDF Value represents the value supplied through the direct SDF file reader. Note that if this table entry is valid, then SDF must be applied during elaboration, to avoid spurious errors.
- Correspondence of SDF Information to VHDL Design, does not adequately describe how to associate SDF constructs with VHDL instances, and there is no example of a small VHDL design and a corresponding SDF file. The distinction between component type and instance of a component is not clearly maintained; the correspondence between cell names and VHDL items (component instance labels? component types? pathnames to what? (p. 96 says concurrent procedure calls too)) is not clear; and the definition of a pathname for VHDL is not provided.
- INTERCONNECT semantics imply a great deal of non-local processing (the identities of the ports can be pathnames, and delays from all drivers to a receiver may need to be checked). Such processing could be very expensive. How difficult, how practical, is this requirement? What about input delays which are not specified in the SDF file -- do they have to participate in the "all drivers to receiver" check?
- Is it required, allowed, or prohibited to use SDF to provide back annotation for VITAL-typed generics on models which are not (entirely) VITAL compliant?
- Generic name for state dependent delay (e.g. IOPATH delay with respect to the value of the input pin). The state dependent delay will be required by most of ASIC foundries. Please standardize the generic names for these IOPATH so that the name can be consistent. And please suggest the delay representation in SDF so that the SDF can be generated correctly.

- SDF mapping method requires the Delay Calculator to read the library data e.g. VHDL model (page 92, appendix C, section 1.2). The Delay calculator will not able to construct the SDF delays from the conditional generic_name. For example: tpd_input_ouput_RESET_EQ_1_AND_CLK_-TI_EVENT_AND_CLK_EQ_1
- The PORT and INTERCONNECT constructs are defined in the same heading, which is confusing.
- Issue: Concern is the perception of which comes first, the VHDL model (ENTITY, ARCHITECTURE) or the SDF file. Obviously, the cell library must exist first, and then designs are developed which use these cells, and finally the SDF file for the design is extracted (computed) from the layout (routing) of the design. This means that the GENERIC parameters on the cell ENTITIES are fixed prior to reading theSDF file. This means that the SDF to GENERIC mapping defines the rules by which constructs in the incoming SDF file are associated with the existing GENERIC parameters on a particular cell. It also implies that there will need to be requirements for reporting errors when the incoming SDF contains delay data which is not supported by the referenced cell. Note: The foundry defines the timing behavior and applicable timing constraints for the cell when it codes the VHDL model. The SDF only fills in the provided parameters.

  Response: In order to standardize on a backannotation mechanism from SDF, so far we have tried to define a mapping between the SDF constructs and the corresponding generic names in the VHDL models (this can be treated as default mapping). However, given that the SDF and VHDL models are produced by different sources, we feel that we also need an additional mechanism (e.g. a mapfile) where the user provides the additional information about the mapping between SDF and VHDL which is applicable to his model/library. This information can then be used by a backannotation program. This mechanism would thus "patch" the problems that could arise because of the SDF generator and the model developer not having information about each other. The mapfile, for example, could contain information whether

  - if a "standard" generic name is not found, which all alternatives should the backannotator look at (in terms of making new generic names), and how.
  - a delay specification for a vector should be uniformaly applied to all bits (i.e. when you have a vector delay path in SDF and scalar paths in VHDL)
  - IOPATH (posedge CLK) Q could be mapped to tpd_CLK_Q in case a generic by the name tEpd_posedge_CLK_Q is not specified in the VHDL model.
  - COND path delays tCpd_CONDMAP_CLK_Q could be mapped to tCpd_CLK_Q if this is the only path between CLK and Q and the generic by the name tCpd.. isnot defined in the VHDL model.
  - a mapping between the default mapping and the "custom" generic names which the user may have used in his VHDL models.
  - generics like tpd_A_Q can be backannotated from

    tpd_A_Q - path delay from A to Q

    tpd_Q - path delay from any input to Q

    tpd - path delay for any path on the part
  - What is the mechanism of guaranteeing the conformance between VHDL model and SDF,given that they are generated by different sources.

Conditional Timing checks:

- Alternate Solution: This solution is based on the assumption that the SDF construct for conditional timing check would be enhanced as shown below. It uses the same scheme as has been outlined earlier for conditional path delays.

  For example, the SDF construct
  COND <condition> SETUP Input1 Input2 (delay)
  would be enhanced to add an optional <condlabel> identified by the keyword VALUENAME as shown below:
  COND <condition> SETUP Input1 Input2 (delay) VALUENAME <condlabel>

  The <condlabel> would be used for identifying the delay value with a certain condition and would be represented in the VHDL model as follows:
  tsetup_Input1_Input2_<condlabel>

Conditional Path delays

- **Limitations of current solution:** There could be potential cases of conflicts when the port names conflict with the names that are generated from the condition expression (by using the above scheme). Since this solution does not take logical equivalence of two expressions into account, two logically equivalent expressions (e.g. A+B and B+A) will result in different generic names. This could cause problems in backannotation in cases where the user has different (although logically equivalent) conditions in the SDF file and his model.

  Alternative Solution:

  In order to take care of the problem mentioned above, we propose an alternative solution which is based on the assumption that the SDF construct for conditional path delays would be enhanced to add an optional <condlabel> identified by the keyword VALUENAME, as shown below:
  COND <condition> IOPATH Input Output (delay)
  would be enhanced to
  COND <condition> IOPATH Input Output (delay) VALUENAME <condlabel>

  The <condlabel> would be used for identifying the delay value with a certain condition and would be represented in the VHDL model as follows:
  tpd_Input_Output_<condlabel>

  - The SDF WRITER/READER would be enhanced to handle the optional <condlabel>. Since the <condlabel> is an optional addition, the new enhanced version of SDF READER/WRITER should be backward compatible.
  - It is assumed that the SDF file is generated (say by a delay calculator) after the VHDL cell (model) has been written. As such, the <condlabel> specified by the user (in his cell) should be used in the generated SDF file.
  - This solution will also be backward compatible with the currently proposed solution (without the additional <condlabel>) because the <condlabel> can be treated as an algorithmically created name instead of being user-defined.