

# An Analysis of the VITAL Initiative

**Victor Berman**

Cadence Design Systems,  
270 Billerica Road  
Chelmsford, MA 01824  
berman@cadence.com

## 1.0 Abstract

The VITAL Initiative was started to promote the availability of VHDL ASIC libraries by providing a solution to the timing/back-annotation problem for VHDL. This paper traces the history of VITAL and analyzed the strengths and weaknesses of the technical proposals which were considered during the design effort. The paper also describes and analyzes the current technical specification which is being handed off to the IEEE Timing Working Group for standardizations.

The analysis focuses on the trade-offs made between flexibility, efficiency, timing accuracy, and industry acceptance. It then discusses the current status and plans for VITAL including the standardization effort and a program for building prototype VITAL libraries.

The major issues dealt with were:

1. Modeling style and timing interface
2. Scope and definition of the underlying timing/delay model and constraints
3. Relationship of specification to related standards (VHDL-92, SDF...)

The paper concludes with a description of the important points in the current Modeling specification with an example of how they are applied to ASIC library cells and a schema for back-annotating those models. The analysis of this example provides insight into their efficiency.

## 2.0 Introduction

The VITAL initiative (VHDL Initiative Toward ASIC Libraries) was started at the 1992 Design Automation Conference with the goal of promoting the early availability of VHDL ASIC libraries. There was general industry agreement that the single factor most responsible for limiting the use of

VHDL in design was the lack of ASIC libraries since the lack inhibited the development of an "all VHDL" path to silicon. Further analysis showed that the major obstacles to library availability was the lack of an industry accepted methodology for modeling timing in VHDL models and for back annotating those models with pre and post-layout timing estimations.

During the months following the 1992 DAC the infrastructure of VITAL developed. A technical lead (Bill Billowitch) and two geographic coordinators (Steve Schulz, North America and Erik Huyskens, Europe) were found. This group held a series of informational and technical meetings to gather requirements and disseminate information about the work being done. The results of this effort was a series of documents beginning with a discussion document which set the agenda for planning followed by a requirements document and several iterations of modeling specifications. At the current time, the 2.1 version of the modeling specification is out for review. It is intended that the result of this review will be an updated document that can be handed off to the IEEE for standardization.

### **3.0 VITAL Phases**

The VITAL effort has evolved in a number of phases beginning with discussion of the goals of the program and going through requirements and specifications. The final phases envisioned are prototyping/validation and standardization. This chapter will outline the history and/or plans for each of these phases.

#### **3.1 Goal Setting**

The goals of this program were set through informal discussions with ASIC vendors, designers, and EDA vendors. The message received was that VITAL should aim at bringing VHDL up to the state of practise enjoyed by ASIC vendors and suppliers using other HDLs such as Verilog and it should be done expeditiously. The message was also clear that timely development was more important than solving new issues in modeling. There was a desire to put the VHDL standard into practice in ASIC design quickly before pressure to get to market forced proprietary solutions to be introduced since this would largely blunt the usefulness of the standard language.

#### **3.2 Requirements Phase**

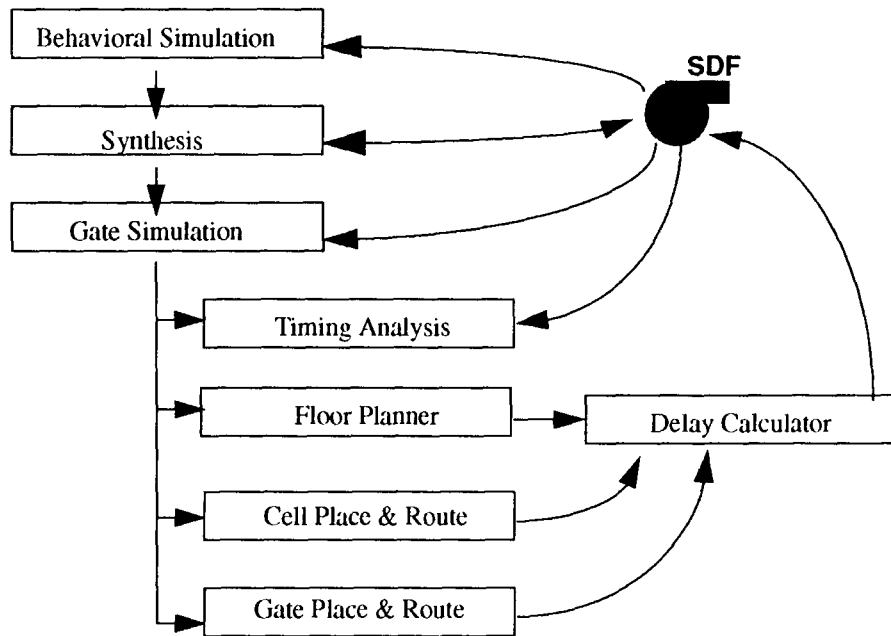
The requirements phase of this project began with series of meetings with ASIC vendors to distill a common set of requirements which would meet the basic needs of a broad set of these vendors. This phase lasted approximately four months and ended with the publication of a VITAL Requirements Specification. Following this, work continued on solving the technical issues introduced by those requirements.

The major requirements identified were:

1. Establish an underlying timing model which provides sufficient accuracy for sign-off simulation.
2. Provide a uniform mechanism for back annotation.
3. Provide a modeling guideline which allows efficient simulation and maps the timing information to VHDL.

### 3.3 Specification Phase

Based on the requirements identified in the previous phase, a Version 1.0 of the VITAL Model Development Specification was released in Feb 1993 for review. This specification introduced several ideas which have remained thematic during the remainder of the VITAL development. First is the idea of the design flow envisioned.



**FIGURE 1. VHDL Based Timing Driven Design**

While no specific design flow is implied, many of the discussions leading to the definition of this specification have centered around a style called Timing Driven Design. In this style information in an external timing file is used to communicate data and constraints between such tools as simulators, synthesizers, timing analyzers and place and route tools. Each tool either uses this data to perform its function or adds to or refines the data.

In addition it was agreed the **SDF (Standard Delay Format)** would be used as the exchange format for timing/back annotation data. SDF is the format which was created to back annotate Verilog and is currently used as an exchange medium among many EDA tools. Adoption of SDF allows VITAL to leverage from this existing work.

VITAL 1.0 adopted a dual approach to modeling ASIC cells.

#### 3.3.1 Macrocells:

ASIC foundries and their customers expressed strong requirements for a solution which will yield high performance. Fast simulation and large capacity are critical issues. Under these conditions, tens and hundreds of thousands of elementary models (i.e. AND gates, flops) need to be simulated quickly and accurately. Rapid turn-around time for circuit changes need to be provided. By separating functional representations from the timing

representations of a model, and by incorporating enhancements within a simulator, significant improvements in runtime speed can be achieved.

### **3.3.2 Megafunctions and complex cells:**

At the complex cell (i.e. 8051 core processor) level, the interaction between behavior and timing rarely lends itself to the techniques outlined for macrocells. Hence, the VITAL program developed an optimized model interface and guideline for the development of complex component models. These VITAL compliant component models have been developed to make use of SDF as do the Macrocell models, but without the added benefit of separation of timing from function as in the macrocell modeling approach.

### **3.3.3 Implicit Optimization of timing (Macrocells):**

In cases where the functionality of a device can be orthogonally separated from the timing behavior of the device (i.e. macrocells), implicit timing semantics can be applied with the objectives of optimizing simulation speed. This technique, already in use by a large number of ASIC foundries within their internal simulation environments, will allow VHDL simulation environments to compete favorably with traditional gate-level simulators.

Explicit Timing Interface (Megacells): In cases where timing and function cannot be readily separated as in complex megacells, back annotation would be explicit and use a timing parameter package to store values. The values in the timing package would be arranged canonically so that they could be indexed efficiently through a simple generic interface. Furthermore, the canonical storage of timing data in the timing package would allow vendors to use a "back door" approach to loading these values directly into the simulator bypassing the expensive process of building and analyzing a large package body or configuration while still maintaining true VHDL semantics in a demonstrable way.

### **3.3.4 Later Changes**

As it turns out, the implicit timing methodology for macrocells was rejected during the review of the VITAL 1.0 specification. It was felt that this method, while very efficient, would require extensive tool changes to implement. It was also felt by some that the idea of implicit timing did fit with the philosophy of VHDL since some of the definition of the model was external and not directly visible in the source text.

Following the review of the VITAL 2.0 specification the idea of the separate timing package was rejected as well. The reasoning here was mainly that the configuration management overhead in synchronizing the separate timing packages with their corresponding models would be too heavy a price to pay for the hoped-for efficiency of this methodology. In its place, the simpler methodology using generics for back annotation was adopted. This methodology had initially been rejected based on poor performance track record, but due to improvements in design of VHDL simulators, this no longer seemed to be a problem.

### **3.3.5 Current Status of Specification**

At the time of the writing of this paper work is ongoing in completing the next draft of the VITAL Modeling Specification. The current schedule calls for the completion of this specification by mid-October 1993. It should contain:

#### **1. VITAL Timing Package**

2. VITAL Primitives Package
3. VITAL Truth/State Table Definition
4. VITAL SDF to VHDL Mapping
5. VITAL Modeling Examples

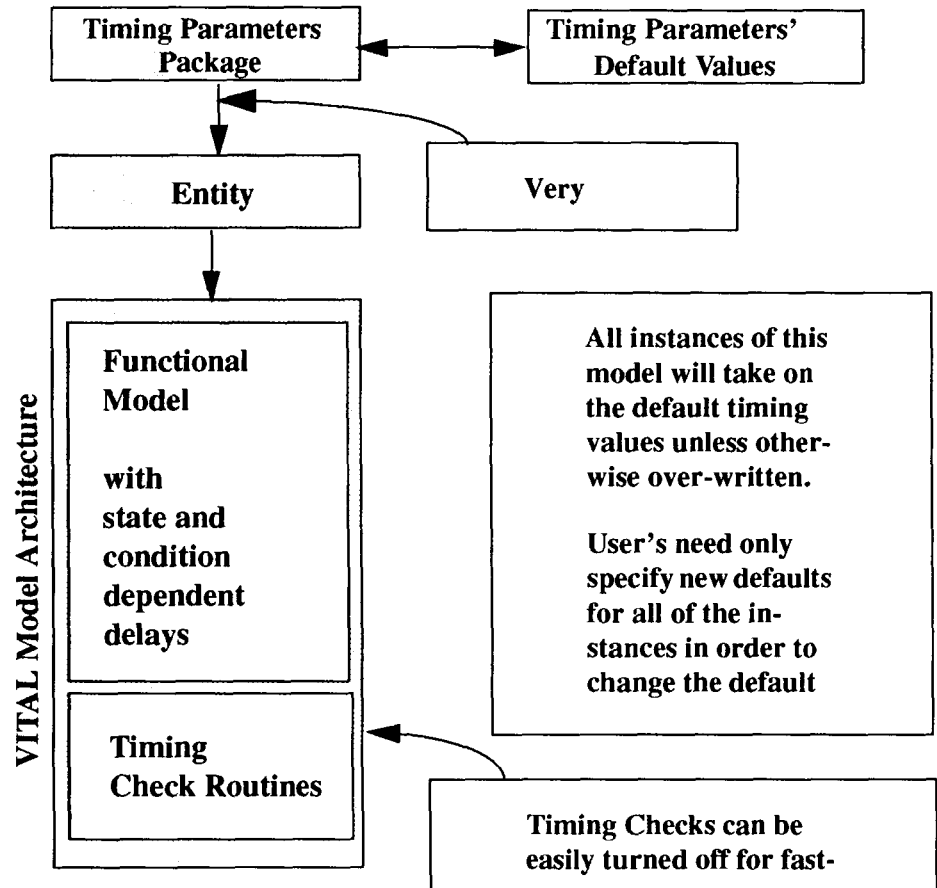


FIGURE 2. VITAL Megafunction Interface Specification

## 4.0 Scope and Definition of Current Development Specification

### 4.1 Overview

The scope of the VITAL Development Specification is to provide the basis for the development of ASIC libraries which meet the requirements for accuracy, efficiency, and ease

of use set in the initial VITAL Requirements Document. Since the beginning of this program the exact meaning of those requirements in terms of level of confidence in libraries developed according to the specification has been hotly debated. Early optimism pointed to the goal of Sign-Off quality libraries. However, as the project progressed it became clear that a specification alone could not guarantee sign-off quality and that the exact definition of ASIC library qualification varied from vendor to vendor. At present the consensus thinking is that the specification can provide a uniform set of conventions, primitives, and timing model which are necessary for developing sign-off libraries, but that further work on modeling guidelines and usage scenarios is needed to be sufficient to guarantee sign-off quality. To accomplish this work VITAL has begun a project for prototyping libraries in a real-life development environment to provide the needed added methodological data to ensure the quality of design and validate the specification.

## **4.2 Timing Package**

Within the current specification work the underlying timing model is best defined by the timing package which contains types, subtypes, functions, and procedures for representing and selecting delays and timing constraints. The body of this package defines the semantics of the functions and procedures in VHDL. Implementations must conform to the semantics defined in the package bodies but may implement them in the most efficient manner possible. The delay selection mechanism is based on a type which enumerates the state transition possibilities. The assumption throughout the specification is that the signal types are based on the IEEE Standard 1164.1 which is the MVL-9 std\_ulogic type. The enumeration covers the common transitions:

```
TYPE TransitionType IS (tr01, tr10, tr0z, trz0, tr1z, trz1);
```

which also follows the convention established by SDF for edge dependence. A function CalcDelay is supplied to determine the proper value of delay to use given the newly assigned and existing value on a signal or driver. Timing check functions and procedures are supplied for setup and hold violations, pulse-width and period checks, and glitch detection. Further details will be supplied at the conference since the specification is not finalized at the time of this writing.

## **5.0 Relationship to Other Standards**

### **5.1 VHDL 1076**

The current charter of VITAL calls for conformance to the VHDL-87 standard. This decision was made to ensure that there was no delay caused by having to wait for VHDL-92 conforming VHDL tools. Several of the VHDL-92 features such as the pulse rejection option on signal assignments and the use of global variable for flag setting/checking would be quite useful. The progress of VHDL-92 tools will be tracked to determine when it will be prudent to begin using such features. Related also is the IEEE 1164.1 Standard for Multi-Valued Logic which forms the basis for the signal type structures used in VITAL.

### **5.2 IEEE Standard Timing Methodology Working Group PAR 1076.4**

This working group has been in operation for approximately one year. It has been working on setting the requirements for VHDL timing/back annotation and has maintained a close liaison with VITAL. This group has agreed to act as the review and analysis body to stan-

standardize the work being developed by VITAL. It is expected that ownership of the Development Specification will be transferred to this group in the October 1993 time frame when it has been satisfactorily approved by VITAL. Furthermore it is anticipated that the VITAL Prototyping Project will continue to work with IEEE 1076.4 during the validation phase so that any changes and improvements in the specification can be rapidly disseminated and tested by the active participants in the Prototyping Project.

### **5.3 Standard Delay Format (SDF)**

SDF is an industry standard notation for exchange delay/constraint data between tools. It is currently controlled by Open Verilog International (OVI), a non-profit group which supports Verilog HDL. SDF was originally developed by Cadence Design Systems to support back annotation in Verilog. The development of SDF was guided by the requirements of many of the ASIC vendors who are now VITAL members and who have been supporting the development of the VITAL Specification. The future control and development of SDF is under debate among the three bodies mentioned above (IEEE 1076.4, VITAL, and OVI).

Currently only a subset of SDF is supported by VITAL. This clouds the issue of support for SDF as an industry standard which includes its use with VHDL. In the author's opinion full support for SDF in VITAL and then by IEEE 1076.4 is crucial for the success of the timing standard and for wide scale adoption of that standard by industry. With the refining and standardization of SDF by IEEE 1076.4 a language independent bridge will be built to semantically connect VHDL and Verilog at the timing data interface. This will drastically reduce the economic burden of library support/maintenance for ASIC suppliers and will lead to the proliferation of high quality ASIC libraries. To facilitate this outcome several technical steps must be taken:

1. SDF definition must be made precise and implementation independent.

Currently the definition is informal and implicitly based on an implementation.

2. A complete mapping of all SDF constructs to VITAL VHDL must be defined.

For clear semantic interpretation this mapping is necessary.

3. A complete mapping of all SDF constructs to Verilog HDL must be defined.

While currently implied, this should be formalized.

### **5.4 CFI Timing, Delay Data, and Constraints (TDDC) Working Group**

This group has been developing an Information Model (IM) for timing information based on the CFI Design Representation, an object oriented data representation which is being developed for frame-work based intertool communication. The TDDC is currently developing an ASCII exchange format based on this IM and is attempting to use SDF as the basis for this format. If successful, this will provide a unifying underlying data model that can be shared by the other users of SDF. This further increases the advantage of VHDL and Verilog both supporting full standard SDF.

## **6.0 Concluding Points**

The goal of VITAL is to accelerate the availability of VHDL ASIC libraries. Initially it was envisioned that this could best be accomplished by directly leveraging from ASIC technology developed for Verilog since this was seen as a de facto standard and was viewed by many as a good solution to the problem. During the evolution of the development specification this idea became somewhat broadened and perhaps blunted. The result is a specification that is perhaps more permissive than is in the best interests of library developers. This has in turn led to the need for a strict set of modeling guidelines within the definition VITAL to further help in the development of efficient, accurate libraries. This function will be filled by the Level-1 Modeling Guide that is developed as part of the prototyping/validation phase of VITAL. This guide is needed for several reasons:

1. Provide a clear, easy to recognize cell based modeling style

Timing must be at the cell level to match the vendors characterization.

2. Provide a clean separation of timing and function in the model.

To enable efficient acceleration clear separation is crucial.

3. Define specific usage scenarios for all SDF constructs.

Without concrete scenarios precise definition becomes complex.

An example using some of the guidelines we envision in Level-1 is shown below:

```
-- SAMPLE ASIC CELL WHICH USES VITAL LEVEL-1 GUIDELINES
-- USING PATH DELAYS
-- It handles
-- - input port delays
-- - accurate pin-to-pin (path) delays
-- - glitch-detection on output
-- - timing checks

LIBRARY IEEE;
USE IEEE.Std_logic_1164.all;
LIBRARY VITAL;
USE VITAL.vital_timing.all;
use work.VITAL_Level1.all;
use VITAL.vital_prim2.all;

-- A simple latch with 2 outputs (q1, q2) and their inverted values (q1n,
q2n)

entity LATCH2 is
    -- Note the naming convention for generics
    -- These can be back annotated
    generic (tpd_data : DelayType01 := DefDelayType01;
            tpd_clock : DelayType01 := DefDelayType01;
            tpd_clock_q1 : DelayType01 := DefDelayType01;
            tpd_clock_q2 : DelayType01 := DefDelayType01;
            tpd_data_q1 : DelayType01 := DefDelayType01;
            tpd_data_q2 : DelayType01 := DefDelayType01;
            tsetup_data : TIME := 1 ns;
```

```

        thold_data : TIME := 1 ns;
        tperiod_clock: TIME := 1 ns;
        tpw_hi_clock: TIME := 1 ns;
        tpw_lo_clock: TIME := 1 ns;
    port (q1, q1n, q2, q2n : OUT std_ulogic;
          data, clock : IN std_ulogic);
end;

architecture PATH_DELAY_VERSION of LATCH2 is
    -- This attribute indicates that model is VITAL LEVEL1 compliant
    attribute VITAL_L1 of PATH_DELAY_VERSION : architecture is TRUE;

    -- A specific naming convention is followed for each temporary
    --signal name
    signal clock_ipd, data_ipd : std_logic; -- delayed input ports
    signal q1_zd, q2_zd, q1n_zd, q2n_zd :
        std_logic;
    -- zero-delay output signals

begin
    -- INPUT PORT DELAY block
    -- Make one call to PropagateInputPortDelay procedure
    -- for each input delay

    INPUT_PORT_DELAY : block
    begin
        PropagateInputPortDelay (clock_ipd, clock, tipd_clock);
        PropagateInputPortDelay (data_ipd, data, tipd_data);
    end block;

    -- FUNCTIONALITY block

    -- This is a zero-delay block. Uses VITAL predefined primitive procedur as
    --concurrent calls.

    FUNCTIONALITY : block
    -- Declare any local signals required to express the functionality here
    begin
        VitalLatch (q1_zd, q1n_zd, data, clock, TimingCheckOn => FALSE);
        VitalBUF (q2_zd, q1_zd);
        VitalBUF (q2n_zd, q1n_zd);
    end block;

    -- Pin to pin delay and glitch handling block

    PATH_DELAY : block

```

```

begin
    -- Make one call to PropagateInputPortDelay for each output
    -- Need a process because we have to save the data for
    -- Glitch handling

    path_q1: process (q1_zd) -- Note the sensitivity list and pro-
process name
        -- Store the information for glitch handling
        -- One per output signal (note the naming convention).
        variable GlitchData_q1 : Glitchdatatype;

        begin
            PropagatePathDelay(q1, "q1", q1_zd,
                -- Make one entry for each path to this output
                ((clock_ipd'last_event, TRUE, ExtendToFillDelay(tpd_-
clock_q1)),
                -- clock->q1
                (data_ipd'last_event, TRUE, ExtendToFillDelay(tpd_data_q1))),
                -- data->q1
                GlitchData_q1, MessagePlusX, 0 ns, OnEvent);
            -- This process should not contain any other statement
        end process;

    path_q2: process (q2_zd)
        -- Store the information for glitch handling
        -- One per output signal (note the naming convention).
        variable GlitchData_q2 : Glitchdatatype;

        begin
            PropagatePathDelay(q2, "q2", q2_zd,
                ((clock_ipd'last_event, TRUE, ExtendToFillDelay(tpd_clock_q2)),
                -- clock->q2
                (data_ipd'last_event, TRUE, ExtendToFillDelay(tpd_data_q2))),
                -- data->q2
                GlitchData_q2, MessagePlusX, 0 ns, OnEvent);
            -- This process should not contain any other statement
        end process;

end block;

-- TIMING CHECKS

TIMING_CHECK : block
begin
    -- We have one process for each timing check. Ideally, the
    -- violation flag should be "shared variable" which can be used
    -- in separate behavioral code to do special processing.

    -- Pulse and Period check

```

```

pulse_clock: process (clock)
    variable PeriodCheckInfo_clock : DelayArrayTypeXX(0 to
3) := PeriodCheckInfo_Init;
    variable violation : boolean := false;
begin
    PeriodCheck (TestPort => clock,
        TestPortName => "clock",
        PeriodMin => tperiod_clock ,
        PeriodMax => tperiod_clock,
        pw_hi_min => tpw_hi_clock,
        pw_hi_max => tpw_hi_clock,
        pw_lo_min => tpw_lo_clock,
        pw_lo_max => tpw_lo_clock,
        info => PeriodCheckInfo_clock,
        violation => violation,
        HeaderMsg => "LATCH2");
    -- Having any kind of behavioral code here will make the recog-
nition
    -- of the timing check functions very difficult. Can this be
avoided?

    -- if violation then
    -- Y <= 'X';
    -- end if;
end process;

timingcheck_data_clock: process
    variable violation : boolean := false;
begin
    violation := TimingViolation (
        TestPort => data,
        TestPortName => "data",
        RefPort => clock,
        RefPortName => "clock",
        t_setup_hi => tsetup_data,
        t_setup_lo => tsetup_data,
        t_hold_hi => thold_data,
        t_hold_lo => thold_data,
        condition => rising_edge(clock),
        HeaderMsg => "LATCH2 ");
    end process;
end block;
end;

```