

# Developing a VHDL Timing Package for Calculating Delays and Timing Constraints

Bihru Tsai  
Intergraph Electronics  
Mountain View, CA

## Abstract

VHDL enables designers to describe their designs in a technology independent format allowing reusability and a smooth migration from one technology to another. To fully realize the benefits of designing with such a methodology, standards need to exist by which component specifications and performance figures may be unambiguously determined and communicated from one technology to another. The establishment of IEEE standard 1164 as a medium for describing the logic states of circuit elements is an example of the type of uniformity that is required. Currently, no standards exist for calculating delays through circuit elements. Each ASIC vendor will most likely have its own set of equations for calculating the delays and timing parameters of their cell libraries. Each vendor will most likely use equations and parameters that will model the actual physical delays in varying amounts of detail. Furthermore, the same parameters are known by different names or have different interpretations. This results in the need for customized packages to handle the requirements of each vendor. If a circuit board is to contain parts from different vendors and they frequently do, a simulation of the board would be a very onerous task requiring the smooth integration of various packages and possibly numerous conversion functions. A universal timing package providing the necessary functions would eliminate such problems. By placing such functions in a package, it is possible for every model to access them and thus guarantee a consistent method of deriving timing data.

This paper discusses the need for, and benefits of, such a timing package. Some guidelines for developing the package are also given. Finally, a sample package implementation is shown.

## Need for a Timing Package

Timing specifications play an important role in the successful operation of any electronic system. The components of a circuit board need to function within a prescribed time window to achieve the desired performance just as the individual components themselves need to be free of any timing errors that arise due to skewing, changing of states, timing constraint violations, temperature or other operating conditions. Unlike functional errors, timing errors are much harder to detect and isolate and are usually triggered by a sequence of events. Developing a design that is free of timing errors involves the cooperation of both the model developer(s) and user(s). The model developer must provide precise detailed information on the timing of the model. The user on the other hand must fully comprehend the timing specifications and assumptions made by the model.

The evaluation of propagation delays through circuit elements is critical in ensuring that the timing specifications of the various devices are satisfied. Since the equations for computing delays are quite complicated and will definitely be used by more than one model, their implementations should be done outside of the model, in a package. Aside from hiding the complex delay calculations from the user, the package also takes away the burden of delay computations and allows the user to concentrate on the functional and timing requirements of the design.

### **Defining a Timing Package**

One of the very first things to consider when defining a timing package is who will use the package and how the package will be used. Once that is established, the requirements and objectives of the package can be drawn. For the timing package under discussion, the users can encompass ASIC designers, board designers, and library developers. The package will be used in evaluating the propagation delays of circuit paths as well as performing timing constraint checks.

The timing package should not assume or impose a modeling style on its user. It should contain all the basic parts that make up the delay equations. It should also take into account the various ways a user may need to calculate delays depending on the circuit topology, and provide functions to meet these needs. The goal here is to create an easy to use, versatile and flexible timing package.

The implementation of the package should be efficient, modular and employ code sharing when appropriate. Meaningful and conventional names should be used as much as possible so that their functionality is intuitive to the user.

All assumptions made during package implementation as well as particular reasons for specific implementations must be clearly and precisely documented so that their interpretation is unambiguous and usage correct.

The timing package should include functions for calculating propagation delays and for determining setup, hold, recovery and minimum pulse width violations. The following factors need to be taken into account when calculating propagation delays :

- a. the intrinsic (pin-to-pin) delay of the device independent of its wiring
- b. the input delay caused by the RC delay and the edge-rate delay at the input pin
- c. the output delay caused by the capacitive load on the driving output pin

If propagation delay is defined as the amount of time it takes a signal to travel through a device and its corresponding interconnect,<sup>1</sup> then the propagation delay from net segment 2 to 3 in the diagram below is :

$$t_{2-3} = t_{RC} + t_{INP} + t_{Pin2Pin} + t_{CL}$$

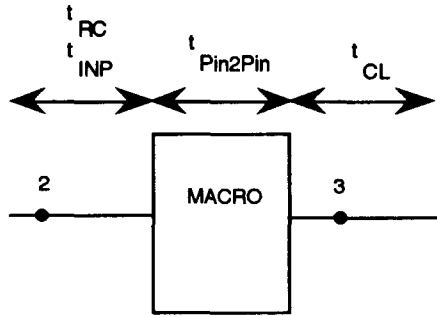


Figure 1. Delay Elements that constitute the propagation delay from net segment 2 to 3

Functions which check that the timing constraints of a device are met also need to be included in the package. Timing constraint specifications typically provide the setup, hold, minimum pulse width, and recovery times required by a sequential device to operate accurately. These timing constraints can be defined as follows :

Setup Time : minimum time a signal (data) must be stable Before the active edge of a reference signal (clock)

Hold Time : minimum time a signal (data) must be stable After the active edge of a reference signal (clock)

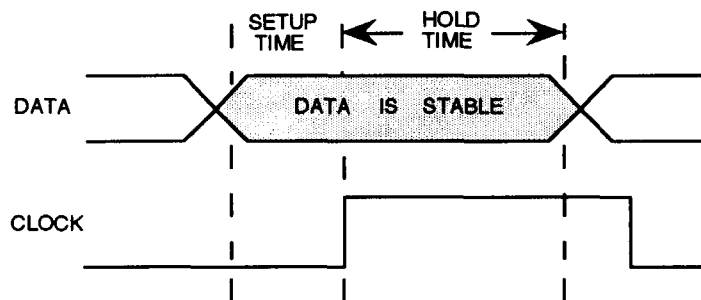


Figure 2. Data Setup and Hold time with respect to rising edge of clock

Minimum Pulse Width : minimum time the reference signal (clock) must remain stable at high or low

Recovery Time : minimum time between the deactivating edge of an asynchronous pin (usually reset) and the active edge of the clock pin<sup>2</sup>

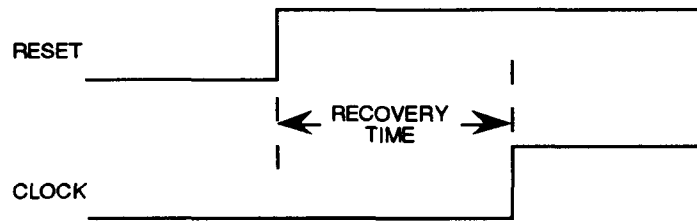


Figure 3. Recovery time between the active rising edge of asynchronous signal reset and the active rising edge of clock

### A Sample Timing Package Implementation

Following is a sample package declaration which employs functions to calculate propagation delays and procedures to perform timing constraint checks. Functions are chosen since there is no need to change the timing parameters, and the derived delay times will most likely be used in an assignment statement or expression. On the other hand, it would be handy to do timing constraint checks in a separate statement in an architecture. The signal parameters passed into the functions and procedures have been declared as type `std_logic`.

The delay equations are taken from the Motorola application note : Delay and Timing Methods for CMOS ASICS. The timing parameters that are provided in a technology file supplied by the vendor are denoted in bold. Since there are similar but separate equations for computing rising and falling delays based on input and output pin transitions, some of the timing parameters also come in pairs, one corresponding to a rising transition and the other a falling transition. Thus, a new type, `vRiseFall` is created to hold a pair of real values and `tRiseFall` a pair of time values.

An effort is made to use meaningful and intuitive names through out the package. The timing and function parameters as well as the operation of the the subprograms are documented in their respective comment sections.

```
-----
library ieee; use ieee.std_logic_1164.all;

package Timing is

type transition is (rising, falling) ;

  -- Type vRiseFall is a duplet of real values. The first one contains
  -- the value associated with the rising edge, the last one for the
  -- falling edge.
  --
type vRiseFall is array (transition) of real ;

  -- Type tRiseFall is a duplet of time values. The first element
  -- contains the value associated with a low-to-high transition,
  -- the latter a high-to-low. For tri-stateable signals, the duplet
  -- can contain values associated respectively with z-to-high and
  -- high-to-z or low-to-z and z-to-low transitions.
  --
type tRiseFall is array (transition) of time ;
```

```

-- Type pmult_array is a triplet of real numbers associated with the
-- bst(commonly known as min), wst(max), typ process multiplier values
--
type procmult is (min, max, typ) ;
type pmult_array is array (procmult) of real ;

-- Process, Temperature, Voltage scaling factor (PTV) calculation :
-- P : procmult = bst(min), wst(max), or typ
-- T : tempmult = y_int + (slope * temperature)
-- V : voltmult = y_int + (slope * voltage)
-- PTV = P * T * V
--
function PTV (proc:procmult; pmult:pmult_array; ty_int,tslope,temp:real;
             vy_int,vslope,volt:real) return real ;

-- Capacitive Load (CL) calculation :
-- Can be used for either input or output load calculation
-- sumIPC : sum of the IPCs (input capacitances)being driven by output
--          pin or being driven on the input net of a cell
-- GATECAP : estimated metal capacitance per connection
-- fan : fanin or fanout, as appropriate
-- SIMLOAD : reference load
-- CLout = sumIPCout + (GATECAP * (fanout + 1)) - SIMLOAD
-- CLin = sumIPCin + (GATECAP * (fanin + 1)) - SIMLOAD
--
function CL (sumIPC:real; GATECAP:real; fan:natural; SIMLOAD:real)
            return real ;

-- Output Load Delay (tCL) calculation :
-- CLout : estimated capacitive load on output pin (from function CL)
-- LoadSlope : (K1,K2) rising/falling loading constants
-- tCLr = K1 * CLout
-- tCLf = K2 * CLout
--
function tCL (LoadSlope:vRiseFall; CLout:real) return tRiseFall ;

-- Input RC delay (tRC) calculation :
-- KRC : RC multiplier (rising,falling)
-- GATERES : estimated gate resistance
-- CLin : capacitive load at input pin (from function CL)
-- tRCr = (GATERES * CLin) * KRCr
-- tRCf = (GATERES * CLin) * KRCf
--
function tRC (KRC:vRiseFall; GATERES,CLin:real) return tRiseFall;

-- Input Edge-rate calculation :
-- T0 : (TR0,TF0) rising/falling no-load output edge-rate
-- EdgeRateSlope : (K3,K4) rising/falling output edge-rate slopes
-- LoadSlope : (K1,K2) rising/falling loading constant of driving cell
-- CLin : capacitive load at input pin
-- tRC : input RC delay (rising,falling)
-- EdgeRate_r = TR0 + K3 * (CLin - SIMLOAD + tRCr/K1)
-- EdgeRate_f = TF0 + K4 * (CLin - SIMLOAD + tRCf/K2)

```

```

--
function EdgeRate (T0, EdgeRateSlope, LoadSlope:vRiseFall;
                  CLin, SIMLOAD:real; tRC:tRiseFall) return vRiseFall ;

-- Input Edge-rate Delay (tINP) calculation :
-- Sense1 : (K7,K8) rising/falling input edge-rate sensitivity factor
-- Sense2 : (K9,K10) rising/falling input edge-rate sensitivity factor
-- TIN0 : input edge-rate used to define intrinsic cell delay
-- TIN1 : input edge-rate at breakpoint of two-piece linear model
--  $tINP_r = K7 * \min(TIN0, EdgeRate_r - TIN0)$ 
--           +  $K9 * \max(0, EdgeRate_r - TIN1)$ 
--  $tINP_f = K8 * \min(TIN0, EdgeRate_f - TIN0)$ 
--           +  $K10 * \max(0, EdgeRate_f - TIN1)$ 
--
function tINP (Sense1, TIN0, EdgeRate, TIN1, Sense2 : vRiseFall)
              return tRiseFall ;

-- Propagation Delay (tProp) calculation :
-- tp2P : intrinsic pin-to-pin delay (rising,falling) values
-- tCL : output load delay (rising,falling)
-- tRC : input RC delay (rising,falling)
-- tINP : input edge-rate delay (rising,falling)
-- PTV : process, temperature, and voltage scaling factors
-- Propagation Delay equations based on macro I/O pin transitions
-- eg. Rr:Rising output,rising input Fr:Falling output,rising input
--  $tPropRr = (TPLHO + tCL_r + tRC_r + tINP_r) * PTV$ 
--  $tPropFf = (TPHLO + tCL_f + tRC_f + tINP_f) * PTV$ 
--  $tPropRf = (TPLHO + tCL_r + tRC_f + tINP_f) * PTV$ 
--  $tPropFr = (TPHLO + tCL_f + tRC_r + tINP_r) * PTV$ 
--
function "tProp" (tp2P, tCL, tRC, tINP : tRiseFall; PTV:real)
                return tRiseFall ;

--
-- Generic Propagation Delay modeling functions
--
-- EdgeType : Can be any two character string where each character is
--             the ascii equivalent of an element in std_ulogic type
-- EdgeDetect : Returns True if a valid EdgeType eg. "10" is detected,
--             and False if an invalid EdgeType eg. "UX" is detected.
-- tProp : Computes the actual value of the propagation delay of a
--         signal. User provides signal and expected direction of
--         change. Function checks that such a change has occurred and
--         returns the appropriate time delay. If the change has not
--         occurred, the returned value is 0 ns.
--
function EdgeDetect (signal S:std_logic; sig_name,EdgeType:string)
                  return boolean;

function "tProp" (signal S:std_logic; sig_name,EdgeType:string;
                 tp2P,tCL,tRC,tINP:tRiseFall; PTV:real) return time ;

--
-- Timing Constraint Checks
--
-- S : signal (usually data) which must satisfy the timing constraints
-- AsyncS : asynchronous signal (usually reset)

```

```

-- clk : reference signal (usually clk)
-- ActiveEdge : active edge of the reference signal (clk) used to
--               perform timing check
-- AsyncSEdge : active edge of asynchronous signal (AsyncS)
-- t : setup, hold, minimum pulsewidth or recovery times to be met
-- SetupCheck : Performs setup time check based on the ActiveEdge of
--               clk. Issues an error if setup time is violated.
--               Will also check for the validity of S being clocked
--               and flag a warning if its state is invalid eg.
--               'U', 'X', 'Z', or 'W'.
-- HoldCheck  : Performs hold time check similar to above.
-- PulseCheck : Performs minimum pulse width check on clk. clk_delayed
--               is a signal derived from clk'delayed(0 ns).
--               Issues error if timing is violated.
-- RecoveryCheck : Performs recovery time check based on active edge
--               of asynchronous signal and active edge of clk.
--               Issues error if timing is violated.
--
procedure SetupCheck (signal S,clk:std_logic; ActiveEdge:transition;
                    S_name,clk_name:string; t:time) ;

procedure HoldCheck (signal S,clk:std_logic; ActiveEdge:transition;
                    S_name,clk_name:string; t:time) ;

procedure PulseCheck (signal clk,clk_delayed:std_logic;
                    clk_name:string; t:time) ;

procedure RecoveryCheck (signal AsyncS, clk:std_logic;
                    AsyncSEdge, ActiveEdge:transition;
                    AsyncS_name,clk_name:string; t:time) ;

end Timing ;
-----

```

## Toward A Standard

Currently, since there are not set guidelines for implementing a timing package, designers and library developers, or anyone who has a need for such a package are writing their own, using proprietary or self defined algorithms. The end result is a series of customized timing packages which contain similar delay functions but are coded differently so that they cannot be readily interpreted.

There is a need for a standard way to represent this timing data so that the information can be consistent, their interpretation unambiguous and their implementation efficient. This paper has attempted to highlight the factors that should be considered when defining a timing package and offer a template for such a package.

VHDL has come a long way since its introduction in 1983 and its acceptance by the design community. It has a growing contingent of experienced users who are aware of the problems and benefits of modeling in VHDL. A standards committee comprising of model developers, semiconductor and EDA vendors should be established and assigned the task of defining and putting together such a standard timing package. By embarking on a standard for calculating delays, model developers, semiconductor vendors and end users can be more productive and gain from the experiences of each other.

## **Acknowledgements**

I would like to express my gratitude to Gabe Moretti for taking time out of his busy schedule to review and discuss the paper with me.

## **References**

1. C. White, "How to Avoid ASIC Timing Problems," ASIC & EDA, August 1992, pp.22-28
2. C. Nakata, "Delay and Timing Methods for CMOS ASICS," Motorola Application Note, 1991