

Timing Constraint Specification in VHDL

Ping F. Yeung

Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, OR 97070

Abstract

We report on how timing constraints can be specified in VHDL for logic synthesis and high-level synthesis. Emphasis is placed on how to present timing constraints so that they can be simulated before synthesis. This helps the user to locate the critical portion of his/her design. Hence, performance constraints can be partitioned accordingly. To do so, a set of timing functions is constructed. They allow *relativity*, *duration* and *event-cause* to be described in various kinds of timing constraints, such as input/output timing constraints and relative timing constraints. Their usages and limitations are discussed.

1 HDL-Based Design

With a hardware description language based design methodology, a design is described at the behavior level. Simulation is carried out to ensure that the functionality of the design is correct. Then, synthesis tools can be invoked to construct the structural implementation of the design. During synthesis, space-time tradeoff is performed under the guidance of the designer. If the behavior description is untimed, no timing information will be available during simulation. This is fine as long as the interface timing is not important. However, to ensure that the performance is right and the timing is correct, system wide timed simulation has to be carried out. It is to make sure that the interface timing of the part is correct and to establish the timing requirement of the design. Therefore, the timing specification will composite of two parts:

1. interface timing specification, and
2. performance timing requirement.

These timing constraints are important as designs always need to interact with the external environment. They must be considered by the synthesis system in order to produce a design which will function properly. The ability of the synthesis tool to meet these constraints is at least as important, if not more, as the quality of the synthesis result. A satisfactory design has to satisfy all timing constraints, design criteria and the interface timing constraints. Currently, timing constraints can only be input interactively or from another constraint file. Including them in the input description certainly helps to provide a clear indication of how the result is achieved. This is also good for documenting the design.

The goal is to enable the description used for system simulation to be used directly for synthesis. Our intention is to construct specifications which can be used for both initial system simulation and synthesis specification. Although some diagnosis statements may need to be commented out, this is not the major issue. The important reason is to preserve the timing specifications in the simulation model.

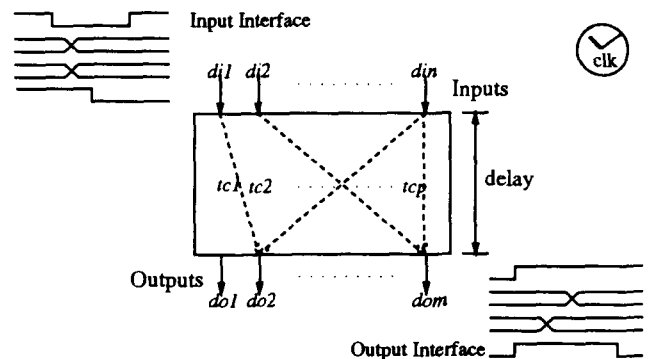


Figure 1: The Timing Behaviors.

The timing considerations of an ASIC design can be effectively summarized as in figure 1.

1. interface timing constraints,
2. overall delay,
3. individual input and output delay,
4. relative path delay,
5. clock synchronization.

The design is in the form of a black box whose behavior can be described at the functional level. The timing constraints specify the input and output signal characteristics that should be exhibited by this black box. The interface timing constraint is the most important. Usually, this is imposed by the timing behavior of the surrounding off-the-shelf chips and modules which will interact with the design. System simulation is performed to iron out any misbehaviour. The overall delay is the value usually used to describe the performance of the design, referred to as the worst case. When multiple inputs and multiple outputs are present, it is necessary to give a more detail individual specification of the I/O ports. Relative path delay details the delay of an individual output delay with respect to changes in individual inputs.

Realization of the timing constraints involves:

- specifying the timing constraints, and
- synthesizing the timing constraints

To synthesize the timing constraints, we first need to specify them. Hence, in this report, we will concentrate on specifying the constraints in VHDL. Research on synthesizing the timing constraints can be found in [2], [3], [5].

2 Timing Constraints

To specify the timing constraints for synthesis, the notion of *relativity*, *duration* and *event-cause* are required [1].

1. Relativity specifies the change of one signal respect to another.
2. Duration specifies the range of delay, say between 20ns to 50ns.
3. Event-cause specifies the characteristics of the event, say signal rising.

There are often situations where we would like to specify constraints such as

```
Bus ← Data (20ns to 50ns) after rising(Ready);
Reg ← PC + 1 (20ns to 40ns) after event(Inc) &
      (10ns to 50ns) after rising(clk);
```

Though VHDL provides support for event-cause using the pre-defined signal attributes, there is no direct support for the other two notions.

3 Timing Functions

To help specify the timing constraints, we first introduce the set of timing functions.

1. *rising_edge*
2. *falling_edge*
3. *time_range*
4. *time_from*
5. *time_from_rising*
6. *time_from_falling*
7. *time_constraints*

3.1 *rising_edge*

Rising_edge(signal) returns **true** when the signal is changing from *low* to *high* in the current simulation cycle; otherwise **false**. It is equivalent to the VHDL expression: (signal'event and signal = '1').

3.2 *falling_edge*

Falling_edge(signal) returns **true** when the signal is changing from *high* to *low* in the current simulation cycle; otherwise **false**. It is equivalent to the VHDL expression: (signal'event and signal = '0')

These two functions are very useful when we want to synchronize some signals with the clock. For example:

```
wait until rising_edge (clock);
y_bus ← data;
```

3.3 *time_range*

This function is to tackle the constraint duration specification.

```
time_range (constraint_degree, lower_time_limit,
            upper_time_limit).
```

It specifies the lower and the upper time limit for a signal to be valid meaning that the constrained signal must be ready between the *lower_time_limit* and the *upper_time_limit*. The parameter, “constraint_degree”, has the type degree which is:

```
type degree is
    (zero, minimum, average, maximum);
```

Referencing to this parameter, the function returns zero, the lower bound, the average, or the upper bound respectively for use in simulation. For example:

```
y_bus <= data after
    time_range (average, 100ns, 200ns);
```

3.4 time_from

The function, *time_from*, is used to specify a relative timing constraint with respect to a particular signal. In figure 4, the timing constraint,

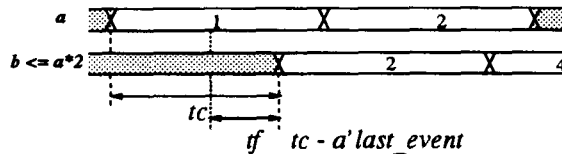


Figure 2: Timing Function: *time_from*.

tc, on the *present signal*, *b*, with respect to the *reference signal*, *a*, can be stated as:

```
b <= a * 2 after time_from (a, tc);
```

$$time_from(a, tc) = \begin{cases} tf & \text{when } tf > 0 \\ 0 & \text{when } tf \leq 0 \end{cases}$$

where $tf = tc - a'last_event$

The function uses the signal attribute, *last_event* to find out how long the reference signal has been stable, and hence calculates how much longer the present signal must wait before it can be activated.

3.5 time_from_rising

The same as *time_from* except it is with respect explicitly to the *rising* edge of the signal.

$$time_from_rising(a, tc) = \begin{cases} tf & tf > 0 \wedge a'last_event = '0' \wedge a = '1' \\ 0 & \text{otherwise} \end{cases}$$

3.6 time_from_falling

The same as *time_from* except it is with respect explicitly to the *falling* edge of the signal.

$$time_from_falling(a, tc) = \begin{cases} tf & tf > 0 \wedge a'last_event = '1' \wedge a = '0' \\ 0 & \text{otherwise} \end{cases}$$

3.7 time_constraints

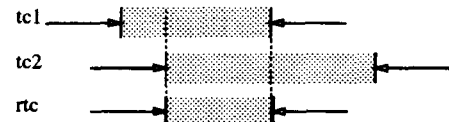


Figure 3: Multiple Timing Constraint.

Time_constraint is the constraint resolution function. It has its functionality similar to a signal resolution function. It takes a vector of time values, then works out the appropriate time value.

```
function time_constraints
    (deg : degree; tv : time_vector) return time;
```

It is usually used with *time_range*.

```
time_constraints (degree, (
    time_range (deg, tc1 ↓, tc1 ↑),
    time_range (deg, tc2 ↓, tc2 ↑)))
```

Referring to figure 3, if “degree” equals “minimum”, *time_range* of the time constraints *tc1* and *tc2* will return their lower bound values, *tc1* ↓ and *tc2* ↓. *Time_constraints*, therefore, should return the maximum of their lower bound values, *rtc* ↓ = max(*tc1* ↓, *tc2* ↓). On the other hand, if “degree” is “maximum”, it should return *rtc* ↑ where *rtc* ↑ = min(*tc1* ↑, *tc2* ↑).

All these functions are defined in the VHDL package listed the appendix. During simulation, with a generic declaration controlling the *degree* of constraint, delay values can be generated to emulate the best and the worst cases.

4 Relative Constraints

Before going into details, let us clarify the difference in specifying overall timing constraint and relative timing constraint. Assume no “wait” statement is present inside the two processes in figure 4. The process (a) is describing

```

process (in1, in2)
begin
    . . . . .
    Out1 ← out1 after tc;
end process
(a)

process (in1, in2)
begin
    . . . . .
    Out1 ← out1 after time_from (in1, tc);
end process ;
(b)

```

Figure 4: Overall Timing Constraint vs Relative Timing Constraint.

the overall timing constraint while the process (b) is describing the relative timing constraint. In process (a), if either signal “in1” or “in2” changes, the output will be delayed by *tc*. For process (b), however, the constraint only applies to changes in signal “in1”. Any change in signal “in2” will be reflected instantly.

A relative timing constraint specifies the changes of a signal with respect to the others. It can be further subdivided into two groups:

1. Relative Path Constraint
if the target signal is data dependent on the reference signal.
2. Relative Event Constraint
if the target signal is controlled by an event happening on the reference signal.

The overall delay is a special case of the relative path constraint which specifies the relative path constraint of all outputs with respect to their data dependent inputs.

4.1 Relative Path Constraint

The relative path constraint has its effect on the the computation delay of the dataflow path. In its general form, relative path constraint of a design can be described as in figure 5. The design is in a process which includes all the input signals in its sensitive list. The output signals are expressed as usual except that the time expression of the after clause is made up by timing functions as detailed in section 3. The functions, *time_constraints* and *time_from* are used. The timing function, *time_from* is used

```

process (in1, in2, . . . in_n)
begin
    Out_i ← out_i time_constraints (degree, (
        time_from (in1, time_range_tc1_i),
        time_from (in2, time_range_tc2_i),
        . . . . .
        time_from (in_n, time_range_tc_n_i)));

    Out_j ← out_j time_constraints (degree, (
        time_from (in1, time_range_tc1_j),
        time_from (in2, time_range_tc2_j),
        . . . . .
        time_from (in_n, time_range_tc_n_j)));
end process ;

```

Figure 5: General Relative Path Constraint.

to specify the relative timing constraint to an input. For each output, timing constraints with respect to each individual input are specified. *Time_range_tc_{ni}* specifies the time range portion of the constraint which is *time_range* (deg, *tc_{ni}* ↓, *tc_{ni}* ↑).

Consider the small example in figure 6. Let’s say, output “q” must be settled within 30ns to 40ns after the value form input “y” has arrived. The constraint on signal “q” will be

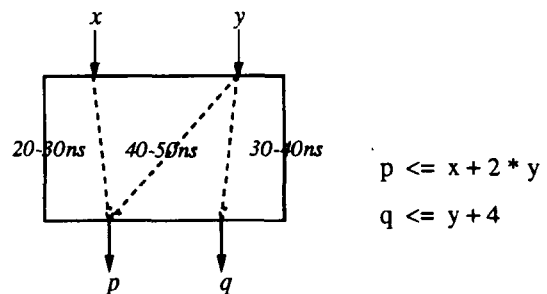


Figure 6: A Simple Relative Path Constraint.

$q \leftarrow y + 4$ after *time_from* (y, *time_range* (deg, 30ns, 40ns));

Again, let’s say, the relative path constraints for output “p” are 20ns to 30ns from input “x” and 40ns to 50ns from input “y”. Since two relative path constraints are involved, the function *time_constraints* must be used to resolve the constraints. It can be expressed as

$p \leftarrow x + 2 * y$ after *time_constraints* (deg, (

```

time_from (x, time_range (deg, 20ns, 30ns)),
time_from (y, time_range (deg, 40ns, 50ns)));

```

It is important to make sure that the timing constraints are realizable. For the above example, if signal “x” and “y” change simultaneously, for output “p”, the two relative path constraints will become disjoint and cannot be satisfied.

4.2 Relative Event Constraint

For relative event constraints, we have *time_from_rising* and *time_from_falling* which are used in the same way as *time_from*. For the timing diagram in figure 7, this can be specified as:

```

Zo <= result_z after time_constraints (deg, (
time_from_falling (c1, tc1),
time_from_rising (c2, tc2)));

```

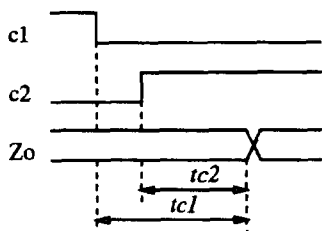


Figure 7: Timing Diagram for Relative Event Constraint.

5 Example

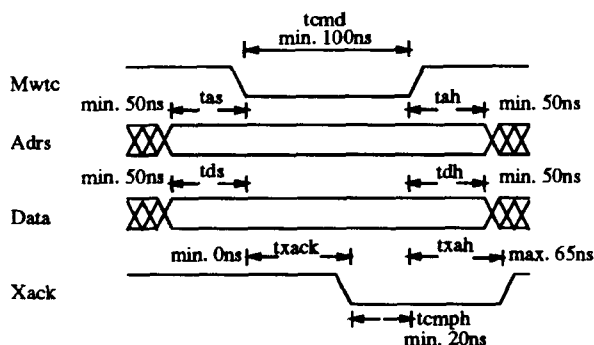


Figure 8: The Timing Diagram of a Bus Write.

In figure 8, we have the Bus Write cycle of a processor. The timing constraints of it is expressed in figure 9. The “Control” process ref-

```

Control : process
begin
wait on Adrs, Data;
Mwtc <= '0' after time_constraints (deg, (
time_from (Adrs, time_range (deg, tas, tmax)),
time_from (Data, time_range (deg, tds, tmax))));
wait until (Xack'event and Xack = '0');
Mwtc <= '1' after time_constraints (deg, (
time_from (Mwtc, time_range (deg, tcmd, tmax)),
time_from (Xack, time_range (deg, tcmph, tmax))));
wait until (Xack'event and Xack = '1');
end process ;

Write : process
begin
wait until (write'event and write = '1');
Adrs <= address;
Data <= data_reg;
wait until (Mwtc'event and Mwtc = '1');
Adrs <= "ZZZZZZZZ" after
time_range (deg, tah, tmax);
Data <= "ZZZZZZZZ" after
time_range (deg, tdh, tmax);
wait Until (Xack'event and Xack = '1');
end process ;

```

Figure 9: Relative Timing Constraints of Bus Write Process.

erences “Adrs”, “Data” and “Xack” to generate the control signal “Mwtc”. In the “Write” process, signals “Adrs” and “Data” are generated with timing synchronized with “Xack” and “Mwtc”. The full description of this example and the simulation result are recorded in the appendix. Generics are defined in the entity declaration to parameterize the timing values. Since, maximum constraints are absent from the original specification, “tmax”, a convenient large value is specified.

6 Limitations

The limitation is associated with relative event constraints. When the event one would like to relate to is not the last event, there are difficulties in expressing it. These events are usually the rising and falling edges which happened two events before. For example, in figure 10, we are interested in the rising edge of the signal “clk” which is not the last event. The last event is the falling of “clk”. In the functions, *time_from_falling* and *time_from_rising*, we use the signal attribute — *last_event* to obtain time lapse after the happening of the last event on the referenced signal. VHDL does not support

the specification of what last event one is interested in and certainly does not have signal attributes such as — *last_rising* or *last_falling*. If we are only interested in one relative event constraint such as figure 10, it can still be described using a process statement like figure 11.

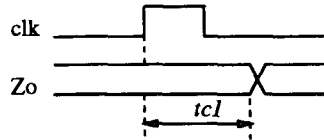


Figure 10: Rising Edge Two Event Before.

```

process
begin
  wait until (clk'event and clk = '1');
  p ← x + 2 * y after time_constraints (deg, (
    time_from (x, time_range (deg, 20ns, 30ns)),
    time_from (y, time_range (deg, 40ns, 50ns))));
end process ;

```

Figure 11: Referencing to a Particular Event.

The main problem happens when one is interested in more than one event constraint. As shown in figure 12, neither the event on signal “c1” nor the event in signal “c2” is the immediate last event. However, if we can synchro-

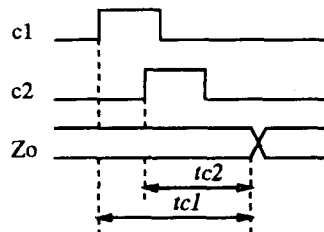


Figure 12: Two Relative Event Constraints.

nize all signals with the clock, this situation is unlikely to occur and would not be a big hindrance.

7 Conclusion

A methodology for describing timing constraints in VHDL for synthesis purposes is reported. This has been proved to be power-

ful enough for describing complex timing diagrams and control/data flow graph with timing constraints. The key is a set of timing function which conform to the VHDL standard and can be interpreted by any VHDL simulator to generate dynamic delay response during simulation. If this set of timing function can be understood by a synthesis system and taken as the flexible timing constraints, then the ideal “what you simulate is what you get” could be achieved. To do so, the synthesis system must be intelligent enough to analysis the performance requirements and to perform timing oriented optimizations.

The restriction imposed by VHDL in referring to a particular last event is also highlighted. With this support, the timing constraints can be expressed more easily.

References

- [1] N.D. Dutt, D.D. Gajski “EXEL: A Language for Interactive Behavioral Synthesis”, Proc. 9th Computer Hardware Description Languages and their Applications, 1989.
- [2] E.F. Griczyc, R.J.A. Buhr, and J.P. Knight “Applicability of Subset of Ada as an Algorithmic Hardware Description Language for Graph-Based Hardware Compilation”, IEEE Transactions on Computer-Aided Design, pp.134-142, April 1985.
- [3] S. Hayati, A. Parker, and J. Granacki “Representation of Control and Timing Behavior with Applications to Interface Synthesis”, Proc. International Conference on Computer Design, pp.382-387, 1988.
- [4] S. Hayati and A. Parker “Automatic Production of Controller Specifications From Control and Timing Behavioral Description”, Proc. 26th Design Automation Conference, pp.75-80, 1989.
- [5] J.A. Nester and D.E. Thomas “Behavioral Synthesis with Interfaces”, Proc. International Conference on Computer-Aided Design, pp.112-115 1986.

8 Appendix

8.1 The Timing Package

```
package Timing is
  type degree is (zero, minimum, average, maximum);
  type time_vector is array(integer range <>) of time;

  function cycle_time (n : integer; cycle_length : time)
    return time;
  function time_max (t1, t2 : time) return time;
  function time_min (t1, t2 : time) return time;
  function time_range (deg : degree; tmin, tmax : time)
    return time;
  function time_from (signal s : integer; t : time)
    return time;
  function time_from (signal s : bit; t : time)
    return time;
  function time_from_rising (signal s : bit; t : time)
    return time;
  function time_from_falling (signal s : bit; t : time)
    return time;
  function time_constraints (deg : degree; tv : time_vector)
    return time;
end Timing;

package body Timing is

  function cycle_time (n : integer; cycle_length : time)
    return time is
  begin
    return (n * cycle_length);
  end cycle_time;

  function time_max (t1, t2 : time) return time is
  begin
    if (t1 > t2) then return(t1);
    else return(t2);
    end if;
  end time_max;

  function time_min (t1, t2 : time) return time is
  begin
    if (t2 > t1) then return(t1);
    else return(t2);
    end if;
  end time_min;

  function time_range (deg : degree; tmin, tmax : time)
    return time is
    variable result : time;
  begin
    if (deg = zero) then result := Ons;
    elsif (tmin >= tmax) then result := tmin;
    else
      case (deg) is
        when minimum => result := tmin;
        when average => result := (tmin + tmax) / 2;
        when maximum => result := tmax;
      end case;
    end if;
    return (result);
  end time_range;

  function time_from (signal s : integer; t : time)
    return time is

    variable result : time;
```

```
begin
  result := t - s'last_event;
  if (result < Ons) then result := Ons;
  end if;
  return (result);
end time_from;

function time_from (signal s : bit; t : time)
  return time is

  variable result : time;
begin
  result := t - s'last_event;
  if (result < Ons) then result := Ons;
  end if;
  return (result);
end time_from;

function time_from_rising (signal s : bit; t : time)
  return time is

  variable result : time;
begin
  if (s'last_value = '0' and s = '1') then
    result := time_from(s, t);
  else
    result := Ons;
  end if;
  return (result);
end time_from_rising;

function time_from_falling (signal s : bit; t : time)
  return time is

  variable result : time;
begin
  if (s'last_value = '1' and s = '0') then
    result := time_from(s, t);
  else
    result := Ons;
  end if;
  return (result);
end time_from_falling;

function time_constraints
  (deg : degree; tv : time_vector) return time is

  variable tmax, tave, tmin, result : time;
begin
  if (deg = zero) then result := Ons;
  elsif (tv'length < 1) then result := Ons;
  elsif (tv'length = 1) then result := tv(1);
  else
    tmax := time'low;
    tave := Ons;
    tmin := time'high;
    for i in tv'range loop
      tmax := time_max(tmax, tv(i));
      tmin := time_min(tmin, tv(i));
      tave := tave + tv(i);
    end loop;
    case (deg) is
      when minimum => result := tmax;
      when average => result := tave / tv'length;
      when maximum => result := tmin;
    end case;
  end if;
  return (result);
end time_constraints;
end Timing;
```

8.2 The Bus Write Interface : Timing Constrained

```

library mcc_primitives; use mcc_primitives.Bit_Types.all;
use work.Timing.all;

entity Multi_Bus is
  generic(tas, tds, tah, tdh : time := 50ns;
    tcmph : time := 20ns;
    tcmd : time := 100ns;
    tmax : time := 1000ns;
    deg : degree := minimum);
  port (Adrs, Data : inout bit_code_vector(7 downto 0);
    address, data_reg : in bit_code_vector(7 downto 0);
    Mwtc : inout bit_code := '1';
    Xack : in bit_code;
    Write: in bit_code);
  begin
end Multi_Bus;

architecture BEHAVIOUR of Multi_Bus is
begin
  process
  begin
    wait on Adrs, Data;
    Mwtc <= '0' after time_constraints(deg, (time_from(Adrs, time_range(deg, tas, tmax)),
      time_from(Data, time_range(deg, tds, tmax))));

    wait until (Xack'event and Xack = '0');
    Mwtc <= '1' after time_constraints(deg, (time_from(Mwtc, time_range(deg, tcmd, tmax)),
      time_from(Xack, time_range(deg, tcmph, tmax))));

    wait until (Xack'event and Xack = '1');
  end process;

  process
  begin
    wait until (write'event and write = '1');
    Adrs <= address;
    Data <= data_reg;
    wait until (Mwtc'event and Mwtc = '1');
    Adrs <= "ZZZZZZZZ" after time_range(deg, tah, tmax);
    Data <= "ZZZZZZZZ" after time_range(deg, tdh, tmax);
    wait until (Xack'event and Xack = '1');
  end process;
end BEHAVIOUR;

```

	inpAddr	inpData	Write	Xack	outAddr	outData	Mwtc
0(0):	00	00	0	1	00	00	1
200(0):	AA	0F	1
200(1):	AA	0F	.
250(0):	0
400(0):	.	.	0
600(0):	.	.	.	0	.	.	.
620(0):	1
670(0):	ZZ	ZZ	.
800(0):	.	.	.	1	.	.	.
1000(0):	B2	47	1
1000(1):	B2	47	.
1050(0):	0
1200(0):	.	.	0
1800(0):	.	.	.	0	.	.	.
1820(0):	1
1870(0):	ZZ	ZZ	.
2000(0):	.	.	.	1	.	.	.