UC Irvine
ICS Technical Reports

Title
Structured modeling for VHDL synthesis

Permalink
https://escholarship.org/uc/item/6v46p1q1

Authors
Lis, Joseph S.
Gajski, Daniel D.

Publication Date
1989-06-11

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Z
699
C3
no. 89-14

Structured Modeling for VHDL Synthesis

by

Joseph S. Lis
Daniel D. Gajski

Technical Report 89-14

Information and Computer Science
University of California at Irvine
Irvine, CA 92717
(714) 856 7063

Abstract: This report will describe a proposed modeling style for the use of the VHSIC Hardware Description Language (VHDL) in design synthesis. We will describe the operations and underlying assumptions of four design models currently understood and used in practice by designers: *combinational logic*, *functional descriptions* (involving clocked components such as counters), *register transfer* (data path) descriptions, and *behavioral* (instruction set or processor) designs. We will illustrate the various uses of the VHDL description styles (*structural*, *dataflow* and *behavioral*) to represent characteristics of each of these design models. Emphasis is placed on how VHDL constructs should be used in order to synthesize optimal designs.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

VHDL [VHDL87] is the IEEE standard language for hardware description. However, the VHDL language does not guarantee uniqueness of descriptions; designs can be described in several ways and at several different levels of abstraction. The process of creating different descriptions is called *modeling*. Unfortunately, models perfectly suitable for one application can be unsuitable for another.

There are three basic application areas: simulation, fault modeling and test generation, and synthesis and silicon compilation. The difference in modeling styles comes from different goals of the application areas.

The goal of simulation is to validate the correctness of the description by measuring output response to input stimuli. Thus, generation of correct values on all signal lines over time is the most important goal. Efficient simulation that simulates only parts of the design where input values are changing is the second goal.

In fault modeling, a fault is injected into the model. This fault is then sensitized and its effects are propagated to an observable output in the description. Sensitization and propagation involves establishing data paths through the description and thus is easier with a structural or dataflow description than with an algorithm.

In synthesis, we are interested in generating a structural description of components from a given library from an algorithmic description. Here, we are interested in

properly connecting all pins on all components instead of observing signal values on some of the pins.

1.1. Motivation

VHDL was designed primarily with one application in mind: simulation. For example, the language allows events that are defined as any change on a signal line through the use of attributes such as QUIET and STABLE. The simulator uses these attributes to detect the occurrence of an event on a signal. Such events are not easily realized in hardware since storage elements are triggered by positive edge or negative edge signal transitions but not both. Similarly, VHDL specifies delay with an **after** clause that does not distinguish among inputs. Efficient synthesis algorithms, on the other hand, require delay specifications for each input-output pair. Furthermore, VHDL guard expressions allow any combination of signals, although designers know that only one signal (clock) is used to trigger writing into storage elements. Designing a register that allows writing by two different clocks is not good design practice.

Since synthesis is becoming more and more important, the trend is to solve the problems of VHDL inadequacies by amending or subsetting VHDL. None of these proposals seem reasonable. As a result of work on the development of the VHDL Synthesis System (VSS) [LisGa88] [LisGa89], we are proposing to develop a modeling style for synthesis that will allow for an efficient generation of high quality designs. This modeling style is similar to the application of structured programming techniques

when using programming languages.

In order to understand our methodology, we will relate design models, description styles and VHDL constructs. First we will look at design models understood and used in practice by designers today. We will describe the operations and underlying assumptions associated with four such models: *combinational logic*, *functional descriptions* (involving clocked components such as counters), *register transfer* (data path) descriptions, and *behavioral* (instruction set or processor) designs.

These design models must be described using the *structural*, *dataflow*, and *behavioral* description styles provided by VHDL. The structural description consists of component declarations, interconnect signal declarations, and component instantiations with port maps. This description style is suitable for describing a captured schematic after a design is completed, and it should be used to describe the design generated by a behavioral synthesis tool.

The dataflow description style is not as closely tied to the actual structural implementation of the design. This description style allows for the specification of concurrent events (data transformations and register transfers) under the control of synchronous (clock) or asynchronous signals. It can be used for combinatorial or functional logic models. The synthesis tool must optimize the design for a given component library. In the case of functional logic, components and connections are shared in time. The machine states are already specified in the description using conventions of the modeling style such as one block statement per state.

Behavioral descriptions are void of any implementation detail. They specify output values in terms of input values over time using an abstract algorithm. A synthesis tool must allocate components, schedule operations into machine states, and interconnect components for these specifications.

The quality of a design as well as the complexity of the synthesis process are directly related to the style of description chosen to represent a particular design model. Certain VHDL constructs or description styles are better suited to describe a particular design model than others. Because VHDL allows the designer several ways of describing the same functionality, it is important to set standard modeling practices for designers using VHDL. These standards should guarantee high quality of synthesized design, while divergence from the standard will result in simulatable but not optimal design.

This report will describe a proposed modeling style for the use of the VHSIC Hardware Description Language (VHDL) in design synthesis. We will describe the operations and underlying assumptions of the four design models identified above. We will illustrate the various uses of the VHDL structural, dataflow and behavioral description styles to represent characteristics of each of these design models. Emphasis is placed on how VHDL constructs should be used in order to synthesize optimal designs.

2. VHDL Design Models

2.1. Design Hierarchy

The *design entity* is the primary hardware abstraction in VHDL. It represents a portion of the hardware design that has well-defined inputs and outputs and performs a well-defined function. A design entity may represent an entire system, a sub-system, a board, a chip, a macro-cell, a logic gate, or any level of abstraction in between. A *configuration* can be used to describe how design entities are put together to form a complete design as shown in Figure 1.

A design entity may be described in terms of a hierarchy of *blocks*, each of which represents a portion of the whole design. The top-level block in such a hierarchy is the design entity itself; such a block is an *external* block that resides in a library and may be used as a component of other designs. Nested blocks in the hierarchy are *internal* blocks, defined by **process** or **block** statements. A structural, dataflow or behavioral description style can be used to express the functionality of an internal block.

Successive decomposition of a design entity into components, and binding of those components to other design entities that may be decomposed in like manner, results in a hierarchy of design entities representing a complete design. Such a collection of design entities is called a *design hierarchy.* The bindings necessary to identify a design hierarchy can be specified in a configuration of the top-level entity in the hierarchy. The design hierarchy concept is illustrated in Figure 1.

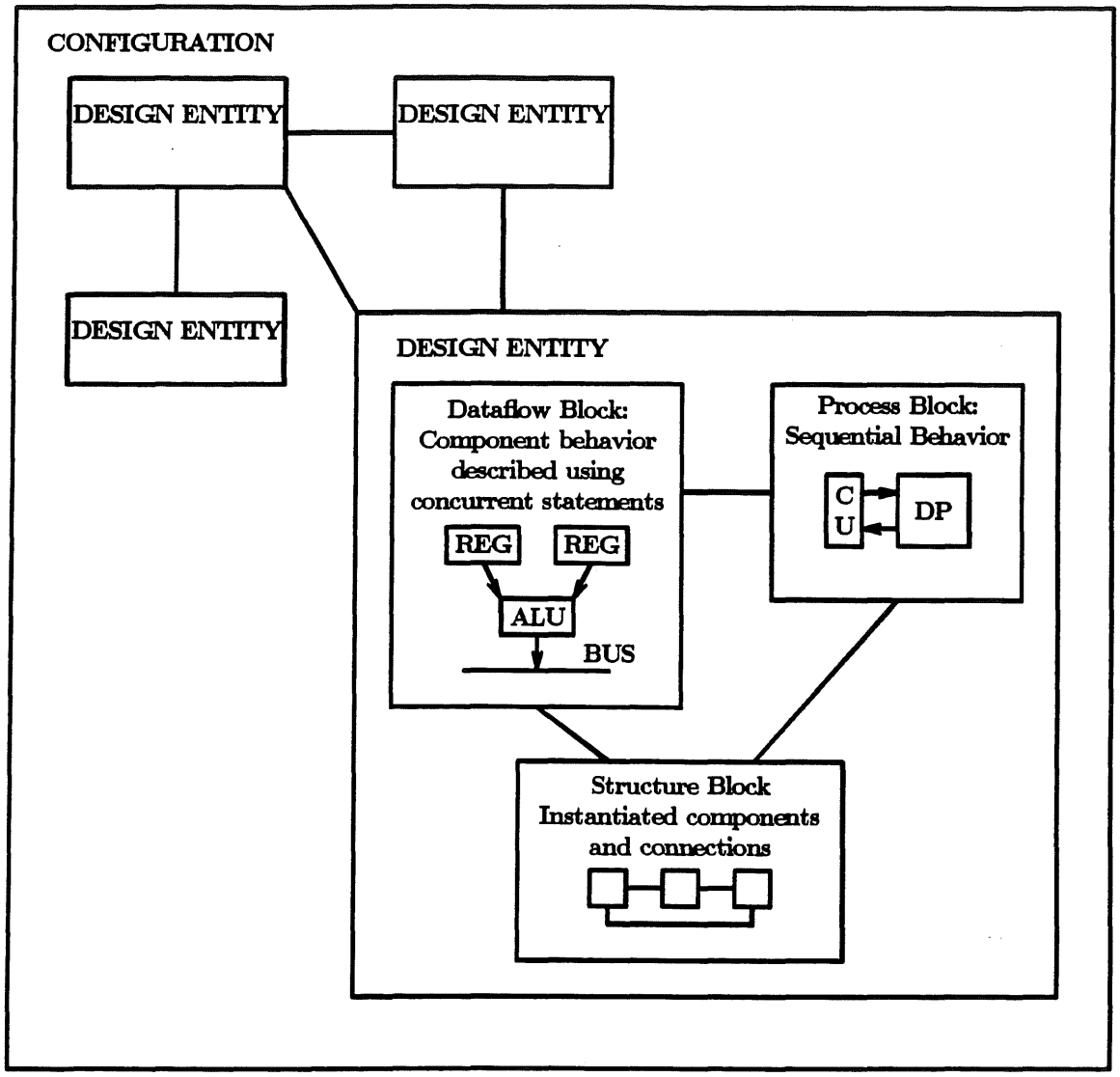

Figure 1: VHDL Design Hierarchy

A VHDL description which represents such a design hierarchy is shown in Figure 2. Each design entity description is composed of two major sections: the *entity block* and the *architecture body*. The entity block contains the specification of external

input/output port connections to the hardware to be designed. The architecture body defines the body (structure and/or behavior) of a design entity. It specifies the

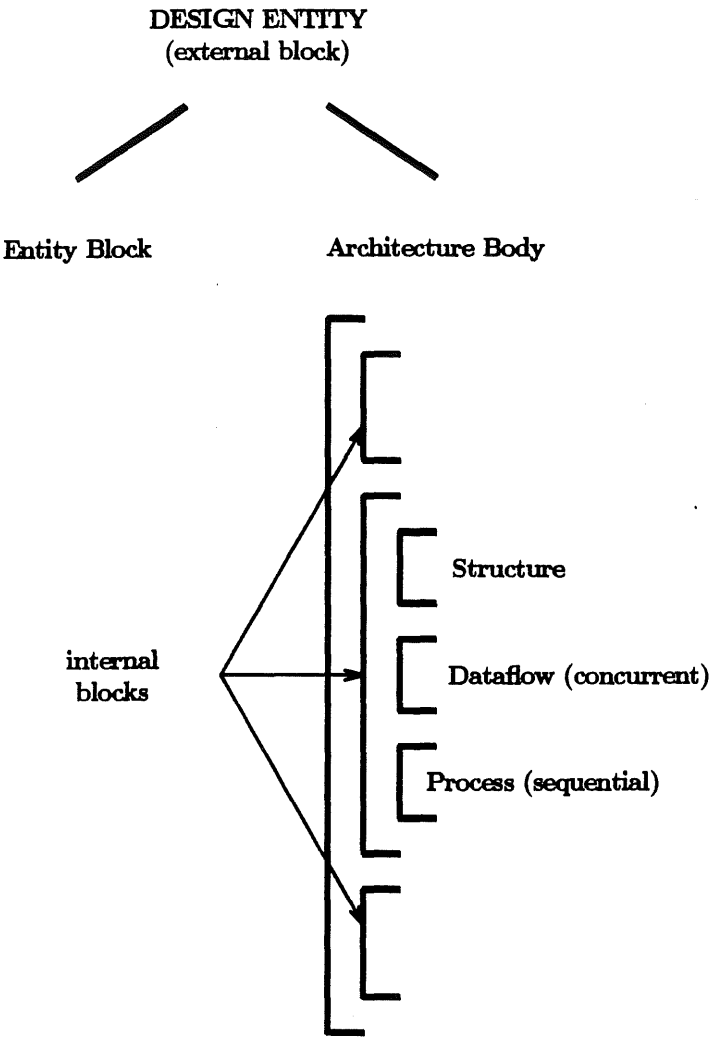

Figure 2: VHDL Design Entity Block Structure

relationships between inputs and outputs of the design entity, and may be expressed using a mixture of the three styles mentioned previously (structural, dataflow, behavioral).

2.2. Design Model

Figure 3 illustrates the underlying design model assumed for a VHDL description [Preas88]. A design is composed of communicating processing elements (PEs). Each PE consists of a Control Unit (CU) and Datapath (DP). Because a process statement may require one or several machine cycles (states) to execute the desired function, the microarchitecture implementation uses the DP to perform computations and the CU to

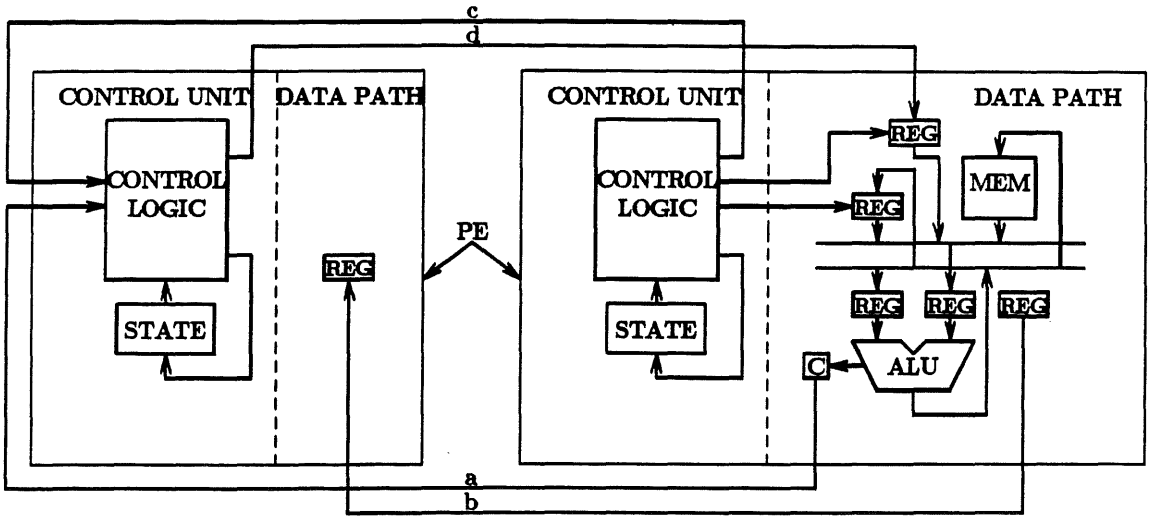

Figure 3: VHDL Design Model

sequence the machine through the necessary states and control the operations performed in the DP for each state. The CU contains a state register for storing the current state of the machine and control logic which controls the DP and communicates with other PEs. The DP consists of storage elements (registers, counters, memories) and functional units (ALUs, shifters, multiplexers) connected through sets of buses.

Access to registers, units or I/O ports is controlled by the CU. If several buses are used as sources to a storage or functional unit, a selector controlled by the CU must be added to the input. Some DP models use only point-to-point connection with selectors only and no buses. Processes also communicate via global signals. PEs communicate through DP ports to the CU or DP (nets *a* and *b* in Figure 3) or through CU ports to the CU or DP (nets *c* and *d*).

Note that in this model, an adder may be represented as a PE with no CU but with a DP (having one output port, two input ports, and no storage elements). Similarly, a flip-flop can be modeled as a DP with no functional units or as a CU with no DP and no control logic. Thus, this model is complete in the sense that it can model any synchronous digital system.

2.3. Design Model Representation

The three description styles (behavioral, dataflow, structural) use concurrent statements to describe a portion of the complete design model shown above. Each concurrent statement in a VHDL description may be used to describe a piece (one or

more components) of a design. Alternatively, more than one statement can be used to describe the functionality of the same design section if the behaviors are non-overlapping (exclusive).

The design sections represented by the concurrent statements communicate via global signals. These signals are defined in the declaration section of the architecture body. A global signal may be read (input) to several blocks or processes, but should be written to (updated by) only one block or process at any given time. In the event that it is desirable to have more than one active driver for a signal simultaneously (to model a bus, for example), a *resolution function* must be written and associated with the signal to determine its proper value for simulation.

2.3.1. Behavior

A VHDL description using the behavioral style consists of *process statements* and *concurrent procedure calls*. Usually, process statements represent programs to be implemented in a microarchitecture which uses the complete control unit/data path design model. Variables within a process may represent storage components or interconnect wires. Local signals are used to communicate between the CU and DP.

Interprocess communication follows these conventions:

(1) The following subtypes are defined for descriptions to be used for synthesis:

```
subtype data is BIT;
```

subtype control **is** BIT;

Signals of type data are used to interface with the data path. Signals of type control interface with the CU.

(2) By default the following signal types/accesses are allowed:

Input
signal/port reads within the data path description
conditional bit signals input to the descriptions of control logic
Output
constant signals output from control logic (boolean, binary, integer)
computed signals output from DP

Timing is expressed as a part of the output signal assignments. Data computations within the process are made with variable assignment statements.

2.3.2. Dataflow

Dataflow descriptions consist of *concurrent signal assignment statements.* They describe only the data path portion of the VHDL design model. The data path is a structure of components, where each component is described by one or more statements.

2.3.3. Structure

The VHDL structural design style utilizes *component instantiation* and *generate* statements. Here, the data path portion of the design model is described through the instantiation and interconnection of component primitives or previously defined design entities.

2.4. Mixture of VHDL Design Styles

This section illustrates a mixture of the VHDL structural, dataflow and behavioral description styles in a single description. Figure 4 shows a block diagram for a controlled counter functional description adapted from [Arms89].

The operation of the controlled counter can be described as follows. On the rising edge of the STRB signal, an internal control register CONREG is loaded with the value on CON. The CONREG value is decoded to perform one of four functions: clear the counter, load a limit register, count up to a limit, or count down to a limit. The counter runs synchronously under an input clock, and the counting functions are enabled by the

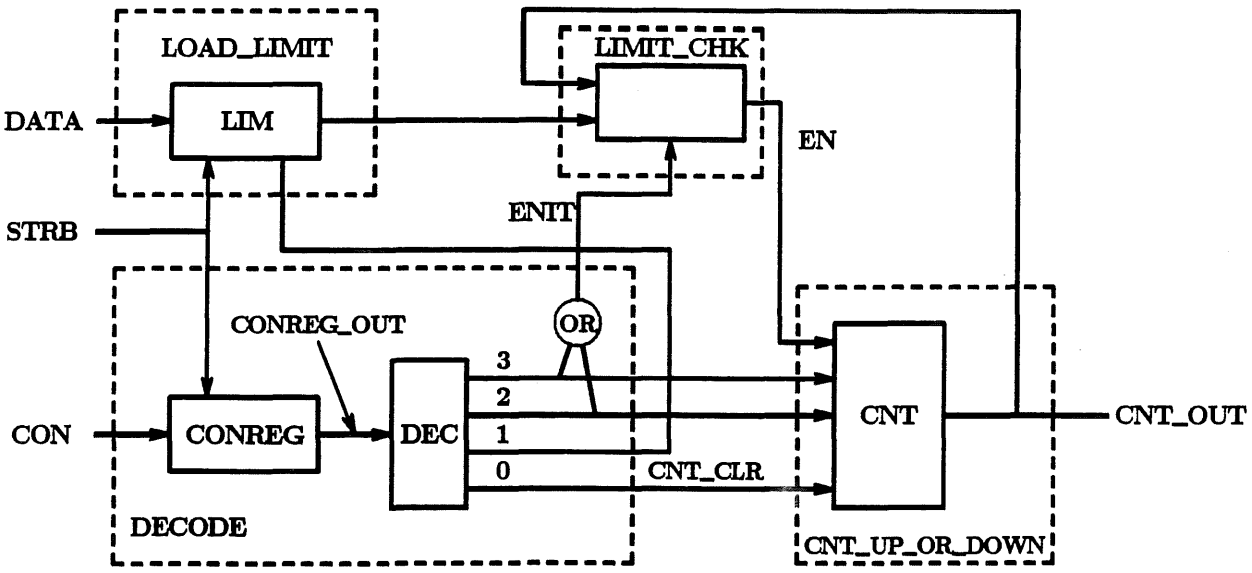

Figure 4: Controlled Counter Block Diagram

internal signal EN. The DATA value is loaded into the limit register LIM on the falling edge of STRB if the control register contains the value '00'.

The VHDL description is shown in Figure 5. This description consists of four block statements, each of which describes a portion of the design: the decoding of the CONREG value, the loading of the limit register (LIM), the asynchronous clear the synchronous up/down count of the counter (CTR), and a limit test.

The DECODE block statement describes the functionality of more than one functional block (the CONREG register and the decoder). A structural description style is used which specifies component declarations, interconnect signal declarations, component instantiations, and component interconnection (via the *port map* clause of the component instantiation statement).

A dataflow description style is used for the LOAD_LIMIT and CNT_UP_OR_DOWN blocks. The block guard is used to enable an update of the LIM and CNT register values. Note that these descriptions carry no information about the structure of the components to be used in the implementation, only the behavior.

The LIMIT_CHK block is described behaviorally with a process statement. This particular description involves only the data path portion of the design model.

```
entity CONTROLLED_CTR is
  port (
       CLK,STRB: in BIT;
       CON: in BIT_VECTOR(1 downto 0);
       DATA: in BIT_VECTOR(3 downto 0);
       CNT_OUT: out BIT_VECTOR(3 downto 0));
end CONTROLLED_CTR;

architecture MIXED of
   CONTROLLED_CTR is

   subtype nibble is BIT_VECTOR(3 downto 0);
   signal CONSIG: nibble := B"0000";
   signal LIM: nibble register := B"0000";
   signal ENIT: BIT := '0';
   signal EN: BIT := '0';
   signal CNT: nibble register := B"0000";
   signal CNT_CLR: BIT;

begin

DECODE: block (STRB = '1')

   component reg
     port (D: in BIT_VECTOR(1 downto 0);
          CLK: in BIT;
          Q: out BIT_VECTOR(1 downto 0));
   end component;
   component decoder
     port (D: in BIT_VECTOR(1 downto 0);
          Q: out BIT_VECTOR(3 downto 0));
   end component;
   component or2
     port (A,B: in BIT;
          O: out BIT);
   end component;
   signal CONREG_OUT: BIT_VECTOR(1 downto 0);

begin

   CONREG: register
     port map (CON,CLK, CONREG_OUT);
   DEC: decoder
     port map (CONREG_OUT,CONSIG);
   OR_1: or2
     port map (CONSIG(2),CONSIG(3),ENIT);
   CNT_CLR <= CONSIG(0);

end block DECODE;

LOAD_LIMIT: block (CONSIG(1)='1' and STRB='0'
        and not STRB'STABLE)
begin

   LIM <= guarded DATA after 10 ns;

end block LOAD_LIMIT;

CNT_UP_OR_DOWN: block ((CLK = '1' and
            not CLK'STABLE) or (CNT_CLR = '1'))
begin

   CNT <= guarded
        B"0000" after 5 ns when CNT_CLR = '1' else
        CNT when EN = '0' else
        CNT + B"0001" after 12 ns
            when CONSIG(2) = '1' else
        CNT - B"0001" after 12 ns
            when CONSIG(3) = '1' else
        CNT;

end block CNT_UP_OR_DOWN;

LIMIT_CHK: process (ENIT,CNT)
   begin

   if ((CNT /= LIM) and (ENIT = '1')) then
     EN <= '1' after 12 ns;
   else
     EN <= '0' after 5 ns;
   end if;

end process LIMIT_CHK;

CNT_OUT <= CNT;

end MIXED;
```

Figure 5: VHDL Description of Controlled Counter

3. Structured Modeling for Synthesis

The quality of a design as well as the complexity of the synthesis process are directly related to the style of description chosen to represent a particular design model. Certain VHDL constructs or description styles are better suited to describe a particular design model than others. Because VHDL allows the designer several ways of describing the same functionality, it is important to set standard modeling practices for designers using VHDL. These standards should guarantee high quality of synthesized design, while divergence from the standard will result in simulatable but not optimal design.

This section describes the design models supported within the VSS system. For each model, the level of abstraction or type of input specification is identified. A VHDL modeling practice for each model is then presented.

3.1. Combinational Logic

3.1.1. Model

The design model for combinational logic consists of a network of logic gates. The most common method used to describe combinational logic designs is boolean equations. In this model, concurrent evaluation of all signal values is assumed. A boolean equation representation facilitates synthesis tasks such as algebraic minimization (e.g., MIS [Bray87]) or optimization (e.g., SilcSyn [BlFo85]).

A combinational logic design involves path delays through the interconnected components. When specifying timing constraints, the combinational logic model should be able to express input to output timing for critical path constraints. These constraints guide the synthesis tool in selecting the appropriate components when tradeoffs are possible. In some instances, the designer may wish to specify more detailed timing constraints on particular operators or paths between some internal points in the design.

3.1.2. VHDL Alternatives

One alternative in VHDL for expressing the combinational logic model is a dataflow description. The combinational circuit can be represented as a set of boolean equations in the form of concurrent assignment statements. Figure 6(a) illustrates a dataflow description of a full adder.

The dataflow description offers the following advantages:

(1) The description style would be familiar to designers who generally think of design at this level in terms of boolean equations.

(2) The description is readable - a straightforward mapping exists between operators and logic components.

(3) In performing synthesis, the description is easily translatable to netlist format (either EDIF or structural VHDL, for example).

Note that timing information is associated with output signal assignments only. If the VHDL description is to remain correct for simulation, timing constraints cannot be specified for internal signals using the after clause mechanism. This is due to the fact that all concurrent assignment statements have their drivers evaluated at the current simulation time using the current value of all signals. Thus, a new value for an internal signal which becomes effective after some delay will not contribute to the computation of a new output value (evaluated at the current simulation time) which depends on it.

An alternative way to describe the functionality of combinational logic is an algorithmic description as shown in the example of the full adder in Figure 6(b).

While expressing the same behavior as the dataflow description, the algorithmic description has the following deficiencies:

(1) The algorithmic description is not the natural way to think of logic. Operators manipulate variables (integers) with extended ranges (number representations) other than boolean. The algorithm requires manipulations of index and other variables. Type conversions from bit quantities to integer and back to perform a counting operation clutter the description and contribute to a suboptimal design generated by the synthesis tool.

(2) Synthesis yields inefficiencies. When the VHDL algorithmic description is used as input for synthesis, the logic that is designed will initially contain some unnecessary hardware. This results from the translation of language constructs

```
entity FULL_ADDER is
  port (X,Y: in BIT;
   CIN: in BIT;
   SUM: out BIT;
   COUT: out BIT);
end FULL_ADDER;

architecture DATA_FLOW_IMPL of
  FULL_ADDER is
    -- local signal declarations
    signal S1, S2, S3: BIT;
  begin
    S1 <= X xor Y;
    SUM <= S1 xor CIN after 3 ns;
    S2 <= X and Y;
    S3 <= S1 and CIN;
    COUT <= S2 or S3 after 5 ns;
end DATA_FLOW_IMPL;
```

(a) Dataflow description

```
architecture ALGORITHMIC_IMPL
  of FULL_ADDER is
begin
  process (X, Y, CIN)
    variable S: BIT_VECTOR(1 to 3);
    variable Num: INTEGER range 0 to 3 := 0;
  begin
    S := X & Y & CIN;
    for I := 1 to 3 loop
      if S(I) = '1' then
        Num := Num + 1;
      end if;
    end loop;
    case Num is
      when 0 => COUT <= '0'; SUM <= '0';
      when 1 => COUT <= '0'; SUM <= '1';
      when 2 => COUT <= '1'; SUM <= '0';
      when 3 => COUT <= '1'; SUM <= '1';
    end case;
  end process;
end ALGORITHMIC_IMPL;
```

(b) Behavioral description

Figure 6: VHDL Full Adder Descriptions

associated with simulator efficiency such as the type conversions mentioned above, or control constructs such as loops which were meant to represent replication of a design section. Additional effort must be spent in the synthesis process to recognize inefficiencies in the design. Some of the inefficiency may never be removed because of costly global optimization.

The following modeling practices for combinational logic are recommended:

Proposition 1

Use the dataflow model for synthesis of combinational logic.

Proposition 2

Use an **after** clause only for assignments made to output signals. This delay represents the maximum allowed delay from any input to the next particular output, and it will be used as a constraint during synthesis.

3.2. Functional Model

3.2.1. Design Model

The functional design model consists of combinational logic as well as storage elements (registers, counters). It may include a mixture of synchronous and asynchronous events for loading storage elements. An event is defined as the transition of a clock or any other signal. It cannot be guaranteed that these events are mutually exclusive; an asynchronous event such as a register reset can occur concurrently with a synchronous load of the same register.

The design is a structure of functional blocks such as ALUs, shift registers, counters, comparators, memories and buses. Each block performs transformations on its inputs with or without latching or storing. Each block is a combinatorial function or a

FSM where the state is determined by the values in storage elements.

The controlled counter [Arms89] shown in Figure 7 is an example of such a design. On the rising edge of the STRB signal, an internal control register CONREG is loaded with the value on CON. The CONREG value is decoded to perform one of four functions: clear the counter, load a limit register, count up to a limit, or count down to a limit. The counter runs synchronously under an input clock, and the counting functions are enabled by the internal signal EN. The DATA value is loaded into the limit register LIM on the falling edge of STRB if the control register contains the value '00'.

The functional design can be described in VHDL using block and process statements. When modeling such a design, one or more functional blocks can be described with one block or process. The counting function of the counter 74LS193 in Figure 7 is described by the block in Figure 8(a). The same function is described by process statement in Figure 8(b).

Modeling each functional block with more than one process may become difficult due to the VHDL limitation of single process assignment. The solution to this problem, proposed by Armstrong [Arms89], is to introduce a virtual multiplexor outside of both processes. This solution, although acceptable in simulation, is difficult to implement in real hardware. Thus, multiprocess modeling of the same functional block should not be used for synthesis.

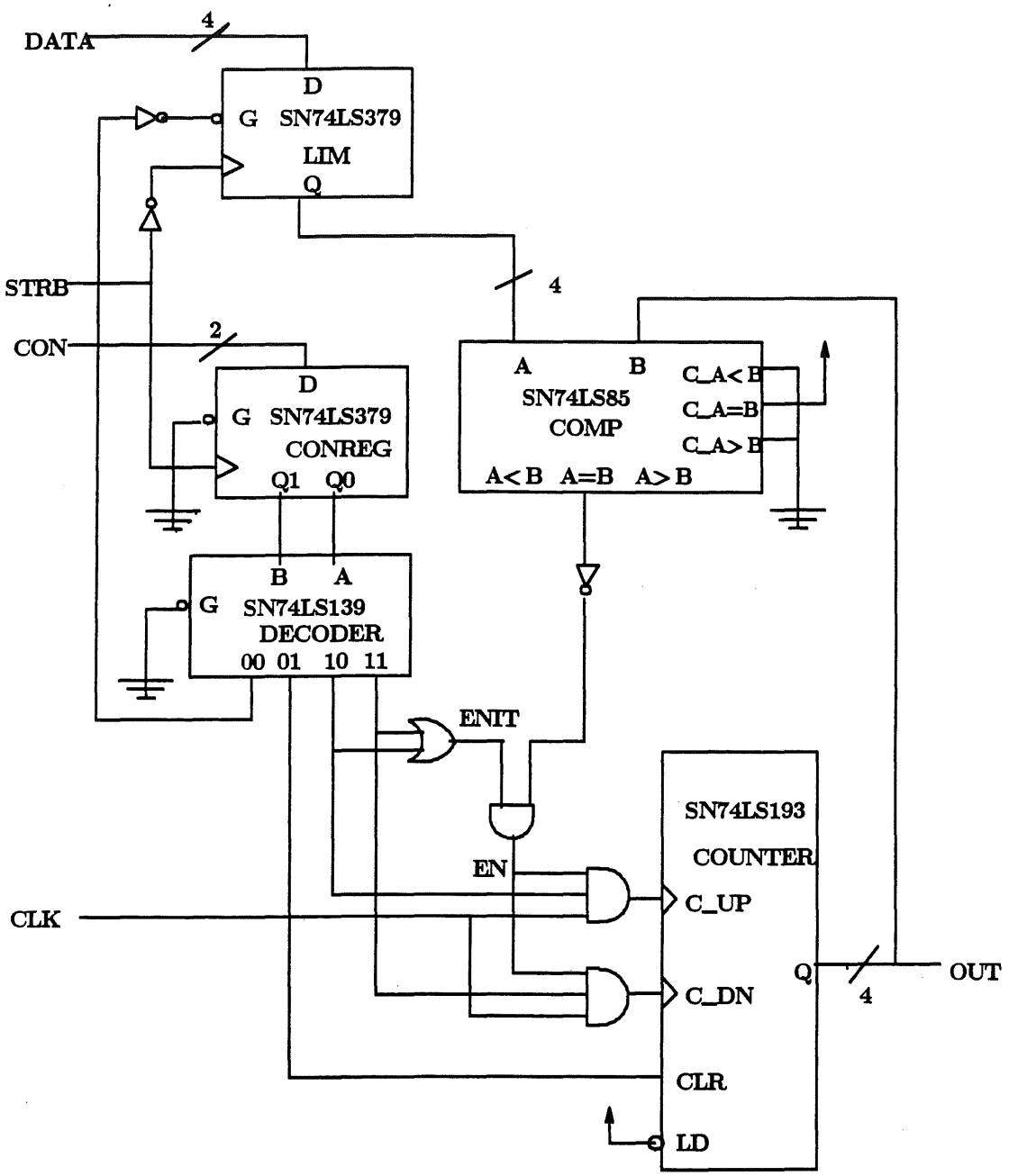

Figure 7: Controlled Counter Schematic

```
CNT_UP_OR_DOWN: block (CLK = '1' and not CLK'STABLE)
begin
    CNT <= guarded
        CNT when EN = '0' else
        CNT + "0001" after INCDEL when CONSIG(2) = '1' else
        CNT - "0001" after INCDEL when CONSIG(3) = '1' else
        CNT;
end block CNT_UP_OR_DOWN;
```

(a) Block Statement Representation

```
CNT_UP_OR_DOWN: process (CLK,CONSIG(2),CONSIG(3),EN)
  variable CNT_REG: BIT_VECTOR(3 downto 0);
begin
  if CLK = '1' and not CLK'STABLE then
    if EN = '1' then
      if CONSIG(2) = '1' then
        CNT_REG := CNT_REG + "0001";
      elsif (CONSIG(3) = '1') then
        CNT_REG := CNT_REG - "0001";
      end if;
    end if;
  end if;
  CNT <= CNT_REG after INCDEL;
end process CNT_UP_OR_DOWN;
```

(b) Process Statement Representation

Figure 8: VHDL Functional Descriptions

Functional blocks can be described with more than one VHDL block statement. However, the behavior described in each block statement should be independent of other blocks. Examples of exclusive functions are the synchronous up counting and asynchronous reset of a synchronous up-counter with asynchronous reset. Furthermore, assignment to the same guarded signal under different guard expressions (representing different clocks) in different VHDL blocks should not be allowed. Although two guard

expressions (i.e., two clocks) can be mutually exclusive, controlling selection of the input signals to the same register may generate timing hazards.

To achieve uniformity, the timing should be assigned only to output signals according to Proposition 2. For each functional block, the following four timing constraints can be used:

a) the clock cycle, specified with a VHDL attribute statement,

b) propagation delay from inputs to (clocked or asynchronously controlled) storage elements. Since this path can contain only combinational logic, a local signal can be defined to designate the storage element input data value. A timing specification (either an attribute or possibly an after clause) can be used for a signal assignment to this local signal.

c) propagation delay from storage elements to outputs, and

d) propagation delay from inputs to outputs (in the case where there are no storage elements on the path from input to output).

In order to properly connect VHDL declared signals to components in the given library, all signals should be typed. The following five types should be defined: **clock, set, reset, test, data** and **control**. Typing will be used to identify the function of event signals appearing in in the block guards. The merging of assignments to the same variable in different blocks is possible during the synthesis process since signal types are known and synchronous/asynchronous behavior is clearly distinguished.

The following guidelines should be followed when developing a functional model description for synthesis:

Proposition 3

One or more functional blocks should be described by one VHDL block statement. Several block statements could be used to describe exclusive behavior (synchronous and asynchronous behavior of the same functional block).

Proposition 4

The guard expression should contain only signals of type clock, set or reset.

Proposition 5

All signals should be typed. Signal types should include clock, reset, set, test, data and control.

3.3. Register Transfer Model

3.3.1. Model

Register transfer descriptions involve the specification of operations to be performed within a PE (as shown in the design model of Figure 3) for each machine state of a design. For each state, one or more triplets specify actions to be performed. Each triplet is composed of a *condition*, a *next state* specification, and a set of

operations. The condition tests a boolean expression. Within each state, one or more conditions may evaluate to true. The actions corresponding to each true condition are performed in the state. If the result of the test is true, a specified set of operations or register transfers is performed. Finally, control is transferred to the specified next state upon completion of the current state operations.

Figure 9 illustrates a simple example of a state table which specifies the conditional statement **if** X = 0 **then** A = A + 1 **else** B = A + B.

Timing in register transfer descriptions is dependent on two things: the specification of a clock cycle, and the maximum time required to perform all operations specified for any state. In this case, it is not necessary to supply timing information in the statements which represent register transfers. If the clock cycle is supplied by the user (using a VHDL attribute for the design entity), the synthesis system will attempt

Current State	Condition	Next State	Ops
S0	True	S1	cond <= (X = 0);
S1	cond	S2	
	cond'	S3	
S2	True	S4	A <= A + 1;
S3	True	S4	B <= A + B;
S4			

Figure 9: Register Transfer State Table

to select units which will perform the desired operations in each state within the specified clock cycle. If no clock is selected, the fastest components are selected from the available library, and the clock cycle is determined by the longest delay path in the design necessary to implement any state.

```
clock_edge <= CLK = '1' and not CLK'STABLE;

State_0: block (clock_edge)
begin
  state <= guarded S1 when (state = S0) else state;
  cond <= (X = '0') when (state = S0) else cond;
end block State_0;

State_1: block (clock_edge)
begin
  state <= guarded S2 when (state = S1 and cond) else
        S3 when (state = S1 and not cond) else state;
end block State_1;

State_2: block (clock_edge)
begin
  state <= guarded S4 when (state = S2) else state;
  A <= guarded A + "0001" when (state = S2) else A;
end block State_2;

State_3: block (clock_edge)
begin
  state <= guarded S4 when (state = S3) else state;
  B <= guarded A + B when (state = S3) else B;
end block State_3;
```

Figure 10: State Table Block Description

In VHDL, block statements may be used to represent the state table using the following conventions:

(1) Every block represents a different state.

(2) The block guard specifies clock, while the body of the block sets the state variable to the appropriate next state and performs operations under the desired conditions.

Figure 10 shows the corresponding block description for the state table of Figure 9. This VHDL block representation allows for the expression of parallelism. Concurrent actions may be specified for a given condition within the block statement.

A second use of the block representation to describe the register transfer state table is shown in Figure 11. This description separates the state transition portion of the description (associated with the control unit) from the register transfers to be performed in each state (data path operations). While this description simulates properly, it has one difficulty from the synthesis perspective: identification of the clock. Assignment to the state variable is made via guarded signal assignments in which the current state, rather than a common clock, is used. The time interval that elapses between changes in the state (the clock period) is modeled with the after clause. The data operations appearing in block statements are also clocked by the state. This description is difficult to synthesize since the clock for register assignments is not explicitly specified.

```
State_1: block (state = S0)
  begin
    state <= guarded S1 after CLK_PERIOD;
  end block;
State_2: block (state = S1 and cond)
  begin
    state <= guarded S2 after CLK_PERIOD;
  end block;
State_3: block (state = S1 and not cond)
  begin
    state <= guarded S3 after CLK_PERIOD;
  end block;
State_4: block (state = S2 or state = S3)
  begin
    state <= guarded S4 after CLK_PERIOD;
  end block;
```

(a) state transitions

```
F0: block (state = S0)
  begin
    cond <= guarded (X = '0');
  end block;
F2: block (state = S2)
  begin
    A <= guarded A + "0001";
  end block;
F3: block (state = S3)
  begin
    B <= guarded A + B;
  end block;
```

(b) data operations

Figure 11: State Transitions/Register Transfers Description

The description of the state table using a process statement is shown in Figure 12(a). Here, each process represents a state. Problems associated with this representation with respect to synthesis include:

(1) One signal variable per state is required. Since each process is triggered by a change in the state variable found in its sensitivity list, detection of this signal change and state decoding are difficult to implement.

(2) The same storage element may need to be updated in more than one process. Using block statements, this can be handled with guarded signal assignments; the

```
architecture P1 of STATE_TBL is
  signal S0,S1,S2,S3,S4,S4_1,S4_2: BIT;
  signal cond: BOOLEAN;
  signal A,A1,A2: BIT_VECTOR(3 downto 0);
begin
  State_0: process (S0)
  begin
    cond <= (X = '0');
    S1 <= not S1 after CLK_PERIOD;
  end process State_0;

  State_1: process (S1)
  begin
    if (cond) then
      S2 <= not S2 after CLK_PERIOD;
    else
      S3 <= not S3 after CLK_PERIOD;
    end if;
  end process State_1;

  State_2: process (S2)
  begin
    A1 <= A + "0001";
    S4_1 <= not S4 after CLK_PERIOD;
  end process State_2;

  State_3: process (S3)
  begin
    A2 <= A + B;
    S4_2 <= not S4 after CLK_PERIOD;
  end process State_3;

  A <= A1 when not A1'QUIET else
       A2 when not A2'QUIET else A;
  S4 <= S4_1 when not S4_1'QUIET else
        S4_2 when not S4_2'QUIET else S4;
end P1;
```

(a) process state description

```
architecture P2 of STATE_TBL is
  type STATE_VAL is (S0,S1,S2,S3,S4);
  signal cond: BOOLEAN;
  signal state,new_state: STATE_VAL;
begin

  process (state)

  begin

    case state is
      ...
      when S0 => cond <= (X = '0');
                 new_state <= S1;
      when S1 => if (cond) then
                   new_state <= S2;
                 else
                   new_state <= S3;
                 end if;
      when S2 => A := A + "0001";
                 new_state <= S4;
      when S3 => B := A + B;
                 new_state <= S4;
      when S4 => ...
    end case;

  state <= new_state after CLK_PERIOD;

  end process;

end P2;
```

(b) single process

Figure 12: Alternative VHDL State Table Descriptions

process, however, provides no clean method of expressing this concept. Variables are local to the process and can be used to represent a storage element within one process only. Guarded signal assignments are not allowed within processes.

Virtual muxes must be added to accommodate the update of the same signal in more than one state. This introduces unnecessary hardware which violates good design practice.

A second use of the process statement to represent register transfers is shown in Figure 12(b). The single process contains a case statement to specify an instruction set like description. This description can't express parallelism for operations associated with one condition since the process is inherently sequential. On the other hand, if we assume for synthesis that all statements appearing within a case alternative are executed in parallel, the VHDL simulation of the input description will not reflect the true behavior of the synthesized design. The solution is to use additional signals of type wire. The following VHDL code fragment illustrates the equivalent sequential statements for the concurrent interchange of the values of A and B:

```
variable A,B: BIT;
signal temp: BIT;

temp <= A;
A := B;
B := temp;
```

In order to describe register transfer designs for synthesis, the following modeling practice is recommended:

Proposition 6

Each state of a register transfer design should be described with block statements containing condition, next state assignment and all register transfers with the clock

specified in the guard expression. Alternatively, a single process with a case statement can be used.

3.4. Behavioral Design

3.4.1. Model

The design model shown in Figure 3 is also assumed for the algorithmic design model. A behavioral description allows the designer to describe the design as a black box with well defined interfaces. Variables within a description can be allocated storage by default, or the synthesis system can determine which variables require storage. As in the combinational model, input to output timing is expressed.

3.4.2. VHDL Description

Figure 13 shows a simple VHDL behavioral description. The process statement is the only suitable method in VHDL for expressing behavior in algorithmic form. Each VHDL process will be synthesized into a CU/DP pair. Data computations within the process are made with variable assignment statements. Its similarity to a programming language allows for the coding of algorithms using typical control constructs (IF, CASE, FOR and WHILE loops).

Input to output timing is expressed as a part of the output signal assignments. The wait statement can be used within the process statement to express timing. A

```
architecture BEHAVIOR of STATE_TBL is
  signal B_port: BIT_VECTOR(3 downto 0);
begin

process (X)
  variable A,B: BIT_VECTOR(3 downto 0);
begin
  if (X = '0') then
    A := A + "0001";
  else
    B := B + A;
  end if;
  B_port <= B after 20 ns;
end process;
end block;
```

Figure 13: Behavioral Description Using VHDL Process Statement

statement of the form

wait until <condition>

will model a design state which loops on itself until the specified condition evaluates to TRUE. The state table entry for this state will advance the state register to the next state in sequence when the condition is TRUE. The second form of the wait statement,

wait for <time>

models a design state which loops on itself for the specified time duration. This model requires a count variable initially set to zero which is incremented on every execution of the state. When the count reaches the specified <time> duration, the state register is advanced to the next state.

The recommended modeling practice for algorithmic design can be summarized as follows:

Proposition 7

Behavioral designs are modeled by VHDL process statements. Signal assignments are used to represent output port assignments. Signals may also be used to hold temporary values (for example, the swapping of register contents) in order to model concurrent events within the sequential process.

4. Conclusion

We have proposed in this report a structured modeling methodology which does not restrict VHDL to a particular subset but recommends several writing styles for different design models. This methodology is based on the following principles:

(1) Appropriate constructs in VHDL should be used for appropriate levels of design.

(2) Guard expressions for block statements are used to represent clocks, or signals that enable storage.

(3) Unguarded signal assignments should be used to model wires. Guarded signal assignments should be used for register and bus assignments. These constructs should not be mixed so that the model remains consistent for synthesis.

(4) Design hierarchy and partitioning should be reflected in the description, although not with the same granularity.

We believe that this structured modeling methodology will make modeling simple, allow portability of models, and facilitate synthesis of high quality designs.

5. References

[Arms89] Armstrong, J., *Chip Level Modeling with VHDL*, Prentice-Hall, 1989.

[Preas88] Preas, B. and Lorenzetti, M., *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, 1988.

[Bray87] Brayton, R., et. al., *MIS: A Multiple-Level Logic Optimization System*, IEEE Trans. on CAD, Nov. 1987.

[BlFo85] Blackman, T., Fox, J., Rosebrugh, C., *The SILC Silicon Compiler: Language and Features*, 22nd DAC, 1985.

[LisGa88] Lis, J. and Gajski, D., *Synthesis from VHDL*, ICCD88, 1988.

[LisGa89] Lis, J. and Gajski, D., *VHDL Synthesis Using Structured Modeling*, 26th DAC, 1989.

[VHDL87] *VHDL Language Reference Manual, Draft Standard* 1076/*B*, IEEE, June 1987.