# Using VHDL for design simulation in a course on digital design with PLDs

André Klindworth
University of Hamburg, Dept. of Computer Science
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany

## Abstract

**This paper presents a course on digital design with programmable logic devices (PLDs) which has been held at the University of Hamburg. In this course, the students design several digital circuits of given functionality using AHDL and Alteras MAX+plusII development tool on SUN workstation. For design validation, the VHDL-simulator *vhdlsim* from Synopsys is used in combination with a tool-set we developped in preparation of the course. This tool-set supports two types of simulation: timing simulation and functional simulation. For timing simulation, the VHDL netlist written by the MAX+plusII compiler is embedded in a testbench ready-to-use with the Synopsys VHDL simulator. For functional simulation, the VHDL model embedded in this testbench is generated directly from the AHDL description, preserving the design hierarchy as well as all symbolic names for instances, signals and machine states as given by the user.**

## 1 Introduction

The programs as described in this paper have been developped in preparation of a course on digital design for programmable logic devices (PLDs). In this course, the students design several digital circuits of given functionality which later on are bound together to form a complex digital system. The circuits are to be realized with Altera PLDs [1] which are dscribed using Alteras hardware description language AHDL and the MAX+plusII development tool [2] on SUN workstations. At the end of the course, an in-circuit test of the design is carried out.

Because of its simplicity, AHDL is much easier to learn than other HDLs like VHDL [4]. As the complexity of PLD-based design projects is limited, AHDL still gives enough expressive power for a compact and straightforward design description. Moreover, AHDL gives the user a better control than standard HDLs over design partitioning and fitting. Both aspects let as prefer AHDL as the means of design entry in a first year graduate course in which most participants have little or no experience in using HDLs as well as PLDs.

Ease of description is one aspect that has to be considered, ease of design verification is another. Validation by simulation is still the usual way verification of digital designs is done. Therefore, the designer should have easy access to a powerful simulator and the development tools of PLD-vendors usually come with programs which generate a design description in a standard format like EDIF, Verilog, and/or VHDL. Such a description then can serve as an input to a third-party simulator.

When simulation is considered, it is highly desirable that the designers view of the design is structured in the same way as in his original description, that is

- design hierarchy,
- groups of signals (busses),
- and all symbolic names of instances, machine states, and external as well as internal signals

should be preserved. MAX+plusII for workstations fails to meet these demands. Netlist writers that produce an EDIF, VHDL or Verilog description of the design are available, but they merely write flattened netlists. Neither the design hierarchy nor names (except for the primary inputs and outputs) are preserved. Functional debugging of a design has to be done by analysis of the primary inputs and outputs only.The netlist writers create netlists that reflect the *physical* structure of the design in terms of logic and interconnect resources of the target PLD. This structure hardly resembles that of the original AHDL description, but allows extraction of actual signal delays. These delays are included in the netlist generated by MAX+plusII so that the user can do a timing simulation of his design using a third-party simulator.

As a compromise between the needs of the designer and what can be fulfilled with a reasonable effort, a tool-set has been developped at the University of Hamburg which eases timing simulation and provides functional simulation of AHDL designs. In both cases, VHDL is used to interface MAX+plusII with the simulator which in our case is the powerful Synopsys VHDL-simulator *vhdlsim* [3].

For timing simulation, the VHDL netlist written by the MAX+plusII compiler is slightly modified and embedded in an automatically generated VHDL testbench. The user may describe his stimuli within this testbench using VHDL, as proposed in [5].

For functional simulation, a VHDL model of the design is generated directly from the original AHDL description. Design hierarchy, busses and all symbolic names of instances, machine states and signals are preserved in this model. It is embedded in the same testbench as the timing model so that the same stimuli are used for both types of simulation. Changing between functional and timing simulation is done by simply configuring the design model that is to be used by the simulator.

This paper presents our course on digital design for PLDs with emphasis on the way simulation of AHDL designs is

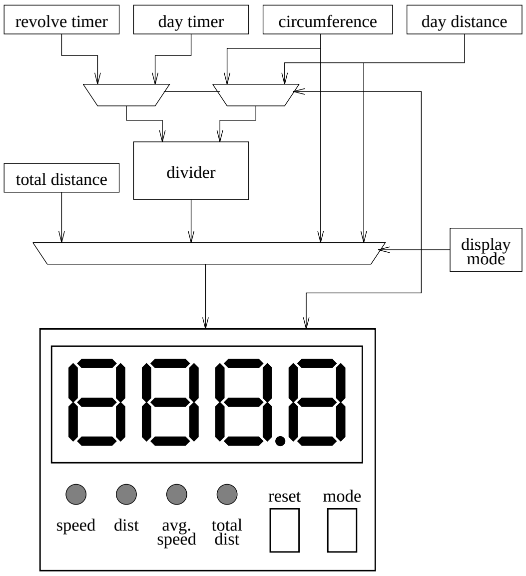


Figure 1: Datapath and user interface of BiCo

done using our tool-set. The next section gives some details on the course and describes the circuits the students design. Section 3 deals with the procedure for doing a timing simulation. After that, the AHDL-to-VHDL compiler *a2vhdl* is presented which generates functional VHDL-model of an AHDL design. This VHDL-model may be used to do a functional simulation of a design project as described in section 5. The paper ends with a summary and some conclusions.

# 2 Course description

Several courses on ASIC design have been held at the University of Hamburg in recent years with full-custom or standard cell ICs as the implementation technology. This course program has now been supplemented by a new course on digital design with PLDs which for the first time gives the opportunity that the students do an immediate in-circuit test of their designs. For this purpose, boards have been prepared which comprise clock generators, buttons, sensors, displays, LEDs, and other surrounding electronic needed by the designs. The students just have to put the programmed PLDs in sockets to do the in-circuit test - perhaps, after a few of the signal connections on the board have been customized for their specific design.

The main design project the students should realize is a bicycle computer named BiCo that measures

- speed,
- average speed since last reset,
- distance since last reset (day_distance),
- and distance since power-up (total_distance).

Figure 2 shows the overall structure of BiCo. The user interface comprises a 7-segment display and two buttons that are used to type in the circumference of the wheel, to reset the design, and to change between the display modes. Revolutions of the wheel are detected using a magnet attached to the wheel and a Reed-switch.

In the first part of the course, the students design and simulate a collection of building blocks like adders, counters, small control units, etc. . These small design tasks demonstrate how the specific structure of the PLDs influences the realization costs and how different design alternatives have to be explored in order to find a good realization under given timing and area constraints.

This first part of the course ends with the design of a traffic light controller that is actually implemented as a PLD and in-circuit tested. This extra design is not part of BiCo and is included as an one-day design task to demonstrate the usage of state machines, synchronisation problems, and the full design procedure from specification to the programmed device.

During the second part of the course, more complicated subcircuits of BiCo are designed using the building blocks from the first part of the course. One of these subcircuits is a divider that divides two BCD-numbers by successive digit-serial subtraction. It is used in the overall design to calculate speeds according to $v = \frac{s}{t}$.

The final days of the course are used to combine the previously designed subcircuits to form the overall design which is simulated extensively. After that, the design is partitioned to be realized with three Altera EPLDs EPM7128LC84 and resource assignments to place the pins and to control the design fitting are added to the design.

# 3 Timing simulation

The procedure for doing a timing simulation of an AHDL design consists of the following steps:

T1: Design entry with AHDL.

T2: Compile design and generate VHDL-netlist.

T3: Modify VHDL-netlist, generate testbench and timing configuration for the design.

T4: Enter stimuli processes in the testbench.

T5: Compile VHDL models.

T6: Start simulator.

The steps T1 and T4 are obviously done by the user using a text editor while step T2 is carried out by the MAX+plusII compiler. A script file called *timegen* realizes step T3: First it checks whether the VHDL-netlist exists and that it is up-to-date. After that, the netlist is modified so that it is compatible with the functional simulation model (see section 5). Among these modifications is a regrouping of busses that have been resolved during compilation. The architecture is named *timing* to distinguish it from the designs functional VHDL model.

Figure 2 shows the VHDL testbench template generated by *timegen*. Its architecture contains a component declaration for the design, a signal declaration for each of its external connections, and an instantiation of the design the ports of which are connected to the signals. Finally, *timegen* creates a file containing a VHDL configuration of the design (figure 3) and a command file for the simulator to trace the primary inputs and outputs of the design instance during simulation.

After the user has added his stimuli processes to the testbench, another script file called *tsim* carries out steps T5 and T6: First it calls the Synopsys VHDL-compiler *vhdlan*

```
ENTITY <testbench> IS
END;

ARCHITECTURE test OF <testbench> IS

 COMPONENT <design>
   <ports of design>
 END COMPONENT;

 <signal declaration for design ports>

-- manual declaration of global
-- procedures, functions and constants

BEGIN

-- manual entry of stimuli-processes

 <instantiation of design>

END;
```

Figure 2: Testbench template

```
CONFIGURATION <cfg_name> OF <testbench> IS
 FOR test
  FOR all:<design>
   USE ENTITY WORK.<design>(timing);
  END FOR;
 END FOR;
END <cfg_name>;
```

Figure 3: Timing configuration of testbench

to generate the simulation models. It then starts the Synopsys VHDL-simulator *vhdlsim* which automatically executes the command file produced by *timegen*.

# 4 AHDL-to-VHDL compiler

At the core of our tool-set is an AHDL-to-VHDL compiler named *a2vhdl* which generates the functional VHDL model of an AHDL design project.

Table 1 shows the correspondence between AHDL and VHDL constructs. To ease the compilation task, *a2vhdl* comes with a VHDL library named `A2VHDL` that contains

- VHDL entities for the AHDL primitives,
- declaration of the types `node` and `node_vector`,
- functions for manipulating values of type `node` and `node_vector`,
- and functions for conversion between type `integer` and type `node` or `node_vector`, respectively.

The new type `node` is used for signals throughout the VHDL model of a design and is declared as a subtype of `std_ulogic`. For this subtype, multiple assignments are resolved by a logical *or* as in AHDL. The functions defined for the types `node` and `node_vector` include logic and arithmetic operators as they are defined in AHDL, e.g. allowing the logical combination of single nodes with signal busses.

| AHDL construct | VHDL construct |
|---|---|
| subdesign | entity |
| subdesign ports | entity ports |
| logic section | architecture |
| primitives | predefined entities |
| function prototype | component declaration |
| node | signal |
| instance declaration | component instantiation |
| inline function call | component instantiation |

Table 1: Correspondence between AHDL and VHDL

```
a <= c * b;
```

is equivalent to

```
a <=  b  WHEN (c = '1') ELSE
     'Z' WHEN (c = '0') ELSE
     'H' WHEN (b = '1') OR (b = 'H') ELSE
     'L' WHEN (b = '0') OR (b = 'L') ELSE
     'X';
```

Figure 4: Conditional operator `*`

In addition, all comparators are overloaded for operands of type `node` to return a value of type `node`. The multiplication operator is not available in AHDL and is overloaded in `A2VHDL` to implement conditional signal assignments according to figure 4. Guarded or conditional signal assignments or a sequential `if`-statement in a process environment could have also been used for this purpose, but would have led to less compact VHDL code, especially, if correct handling of undefined signals is desired.

Figure 5 shows a small AHDL design example and the corresponding VHDL code generated by *a2vhdl* is depicted in figure 7. The declaration of the primitive `dff` is found in the package `A2VHDL.components` and is shown in figure 6. Note the conditional assignments to the data inputs of the register and the way connections to ports of instances are made.

The functional VHDL model of an AHDL state machine consists of the following items:

- Enumeration type declaration for the states of the state machine
- Signal declaration for present state, next state and the control inputs of the state machine.
- Signal declaration for the machine bits if given in the AHDL description.
- Function definitions for testing the FSMs state.
- Process for realizing the state transitions and machine bits assignments.

# 5 Functional simulation

The procedure for doing a functional simulation when using *a2vhdl* consists of the following steps:

```
SUBDESIGN timer (
 clk, clear: INPUT;
 is_zero: OUTPUT;
)
VARIABLE reg[1..0]: DFF;
BEGIN
 reg[].clk = clk;
 IF clear
   THEN reg[] = 0;
   ELSE reg[] = reg[] +1;
 END IF;
 is_zero = (reg[] == 0);
END;
```

Figure 5: An AHDL design example

```
 COMPONENT dff
  PORT (
    d, clk, clrn, prn:  IN node;
    q: OUT node
  );
 END COMPONENT;
 TYPE VHDL_dff_TYPE IS node_vector(1 to 5);
 CONSTANT VHDL_dff_DEFAULT:
            VHDL_dff_TYPE := "XX11X";
```

Figure 6: Declaration of primitve dff

```
ENTITY timer IS
 PORT (clk, clear: IN node; is_zero: OUT node);
END timer;

ARCHITECTURE functional OF timer IS
 SIGNAL VHDL_reg0:VHDL_dff_TYPE:=VHDL_dff_DEFAULT;
 SIGNAL VHDL_reg1:VHDL_dff_TYPE:=VHDL_dff_DEFAULT;
 SIGNAL N_1, N_2: node_vector(1 to 2);
BEGIN
 reg1: dff PORT MAP (
        VHDL_reg1(1), VHDL_reg1(2), VHDL_reg1(3),
        VHDL_reg1(4), VHDL_reg1(5)
      );
 reg0: dff PORT MAP (
        VHDL_reg0(1), VHDL_reg0(2), VHDL_reg0(3),
        VHDL_reg0(4), VHDL_reg0(5)
      );
 reg <= VHDL_reg1(5) & VHDL_reg0(5);
 VHDL_reg1(2) <= clk;
 VHDL_reg0(2) <= clk;
 N_1 <= clear * "00";
 VHDL_reg1(1) <= N_1(1);
 VHDL_reg0(1) <= N_1(2);
 N_2 <= (NOT clear) * (reg + "01");
 VHDL_reg1(1) <= N_2(3);
 VHDL_reg0(1) <= N_2(4);
 is_zero <= (reg = B"00");
END functional;
```

Figure 7: VHDl description generated by *a2vhdl*

F1: Design entry with AHDL.

F2: Compile design with MAX+plusII.

F3: Do AHDL to VHDL conversion for each subdesign of the design project; generate testbench and functional configuration for the design.

F4: Enter stimuli processes in the testbench.

F5: Compile VHDL models.

F6: Start simulator.

Steps F1 and F4 are identical to T1 and T4 of the timing simulation, respectively. In particular, both types of simulation use the same testbench in which the user integrates his stimuli processes. The script file that realizes step F3 is named *funcgen*. After it has checked that compilation in MAX+plusII succeeded, it calls *a2vhdl* for each subdesign that belongs to the design project. To distinguish the functional VHDL model from the timing model, the architecture *a2vhdl* generates for the design is named *functional*. *funcgen* determines the subdesigns of the design project by reading the hierarchy information files (*.hif*) produced by the MAX+plusII compiler. If no testbench exists from a previous run of *timegen* or *funcgen*, *funcgen* also generates the testbench template shown in figure 2. Finally, *funcgen* creates the functional VHDL configuration of the design and the simulator command file to trace the primary inputs and outputs of the design instance.

Steps F5 and F6 are carried out by a script named *fsim* which makes the calls to the VHDL-compiler and the simulator.

The VHDL code generated for the design is transparent. What is solely important to the user is the property that the VHDL model is structured in the same way as the original AHDL description of the design. All design objects the user has used in the original design description do exist in the VHDL model and internal signals may be traced during simulation using the original (hierarchical) node identifiers.

# 6 Conclusion

The tool-set as described in this paper proved to greatly simplify debugging of PLD designs. It combines the compactness of design description in AHDL with the expressive power of VHDL for the description of simulation stimuli and the monitoring capabilities of a powerful VHDL simulator.

# References


[1] *Altera Databook*, Altera Corporation, 1993.

[2] *Altera MAX+plusII reference manual*, Altera Corporation, 1992.

[3] *Synopsys VHDL Simulator Reference Manual*, Synopsys Inc., 1993.

[4] *IEEE Standard VHDL Language Reference Manual*, IEEE Standard 1076-1987, New York, 1988.

[5] Z. Navabi: *VHDL - Analysis and Modelling of Digital Systems*, McGraw-Hill, New York, 1993.