A Platform for Visualizing Digital Circuit Synthesis with VHDL

Abdulhadi Shoufan
shoufan@iss.tu-darmstadt.de
Zheng Lu
zheng@iss.tu-darmstadt.de
Guido Rößling
roessling@acm.org
Technische Universität Darmstadt
Dept. of Computer Science
64289 Darmstadt, Germany

ABSTRACT
This paper presents the VISUAL-VHDL platform for visualizing digital circuit synthesis based on the hardware description language VHDL. VISUAL-VHDL enables students to enter VHDL code and control an animation process, which shows step-by-step how the different language constructs are treated to synthesize a complete digital circuit. It also enables the visualization of the Quine-McCluskey algorithm, which is embedded in our tool to optimize the circuit resulting from synthesizing the VHDL code. The drag and drop schematic editor for entering and parameterizing digital circuits can be used by educators and students to effectively produce circuit diagrams.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer & Information Science Education - Computer Science Education; B.6.3 [Logic Design]: Design Aids - VHDL

General Terms
Management

Keywords
animation, digital circuit synthesis, VHDL, visualization

1. INTRODUCTION
Digital logic is a core unit in most curricula of computer science and electrical engineering. A conventional course on this subject addresses topics such as Boolean algebra, logic gates, optimization of logic circuits, combinatorial logic, sequential logic, registers, counters, arithmetic units, and memories. Traditionally, these topics are introduced using schematic symbols for the different elements of the digital circuit. Several tools were presented to support students in learning logic design using schematic-based simulators [5, 9].

With the ever increasing density of integrated circuits, schematic editors have long become inappropriate for design entry and are replaced by hardware description languages (HDLs) such as Verilog and VHDL. These languages are mostly taught in advanced courses on the design of digital circuits and systems for students of computer or electrical engineering [4]. To expose students to HDL in an early stage of their study, some instructors even start teaching its basics along with the foundations of digital logic.

VHDL is a well-established language for hardware description in industry and research and has gained considerable attention in education [8, 1, 2]. An efficient learning of VHDL relies on understanding it as a hardware description language and on a clear differentiation between this language and traditional programming languages such as C.

Writing a software program mainly consists of understanding the problem, translating the specification into the language syntax, and testing the written program. Questions relating to resource allocation, mapping of tasks to resources, or scheduling are not of interest, as a rule. This is attributed to the fact that most applications today still use only one central processing unit —although two or more cores are common in current computers— which makes the problems of resource allocation and task mapping irrelevant during software programming. Due to the sequential operation of these CPUs, furthermore, the scheduling task is also trivial.

Using VHDL to model digital systems, in contrast, is highly different. Besides understanding the system specification, knowing the language syntax, and the need to test models by simulation, VHDL designers are responsible for solving all the above problems of allocation, mapping, and scheduling, along the way—or perhaps essentially. Understanding VHDL relies on the awareness that writing any VHDL statement may be associated with allocating a new hardware resource, mapping some task of the system specification to this resource, and scheduling the execution of this task at a specific point in time or in a specific clock cycle.

To make students familiar with its basic concepts, we created a visualization and animation platform for VHDL. With the aid of this tool, denoted as VISUAL-VHDL, students can enter small VHDL codes and control an animation process to see how this code is processed to set up the corresponding digital circuit compounded of basic logical elements such as gates, flip-flops and multiplexers.

Table 1 contrasts the differences between VISUAL-VHDL and the schematic viewers embedded in commercial synthesis programs which are caused by our educational design.
2. UNDERLYING CONCEPTS

2.1 VHDL Overview

VHDL (see, e.g., http://www.eda.org/vhdl-200x) is a language for hardware description, simulation, and synthesis, which is standardized by IEEE. The first version appeared 1987 as IEEE 1076-1987, the most recent version in 2008 as IEEE 1076-2008. A VHDL model contains at least one *entity* and one *architecture*. The entity describes the interface of a hardware module. The architecture describes its behavior and/or its structure. Listing 1 shows a simple VHDL model for a synchronous half adder. Six ports are declared in the entity: four input signals (the summands $A$, $B$, a reset signal $RES$ and clock signal $CLK$) and two output signals. In the architecture declaration part, two internal signals $S1$ and $S2$ are declared. Note that all signals are of type bit, i.e., they accept only the values 0 or 1.

The architecture contains two concurrent statements, which assign the sum and the carry to the internal signals $S1$ and $S2$, respectively. The process describes the clock-driven transfer of the signals $SUM$ and $CARRY$. The signals listed after the keyword *process*, i.e., $RES$ and $CLK$, build a so-called sensitivity list: the process is activated upon an event on any of these signals. The condition ($CLK’event and CLK=’1’) monitors the rising clock edge. Synthesizing this process would infer two flip-flops with a reset input.

<table>
<thead>
<tr>
<th>Commercial Schematic Viewers</th>
<th>VISUAL-VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schematic is generated after completing the synthesis process, which often takes a considerable amount of time.</td>
<td>Schematic is generated on the fly.</td>
</tr>
<tr>
<td>Schematic is output all at once. Understanding which VHDL statements were mapped to which schematic elements often demands an accurate investigation of the schematic and the VHDL code.</td>
<td>Schematic is generated dynamically in an interactive mode. During this animation, relating VHDL statements and schematic elements are highlighted using colors.</td>
</tr>
<tr>
<td>Investigation of relation between the VHDL code and the schematic often demands a switching between the windows of the VHDL editor and the schematic viewer.</td>
<td>Both the VHDL code and the schematic are displayed in one window, which considerably facilitates analysis.</td>
</tr>
<tr>
<td>Code optimization may hinder the understanding of the mapping process of the VHDL code to hardware elements.</td>
<td>Code optimization is done on demand. The user can switch the optimization option on or off.</td>
</tr>
<tr>
<td>Code optimization is performed in the background.</td>
<td>Code optimization can be visualized in an auxiliary window.</td>
</tr>
</tbody>
</table>

**Table 1: Commercial Schematic Viewers vs. VISUAL-VHDL**

**Listing 1: A Half-Adder Model in VHDL**

```vhdl
1 entity HALF_ADDER is
2 port( signal A, B, RES, CLK : in bit;
3 signal SUM, CARRY : out bit);
4 end entity HALF_ADDER;
5
6 architecture BEHAVIOR of HALF_ADDER is
7 signal S1, S2 : bit;
8 begin
9 S1 <= A xor B;
10 S2 <= A and B;
11 process (CLK, RES)
12 begin
13 if RES='1' then
14 SUM <= '0';
15 CARRY <= '0';
16 elsif (CLK’event and CLK='1') then
17 SUM <= S1;
18 CARRY <= S2;
19 end if;
20 end process;
21 end architecture BEHAVIOR;
```

**Listing 2: Example for AnimalScript**

```animal
1 triangle "d1" (25,100) (25,110) (55,110)
2 polyline "p0" (35,0) (35,20) (85,20) (85,90) hidden
3 move "d1" via "p0" within 3000 ms
```

Animation platforms for data structures and algorithms in terms of pseudo code or software programs have long been used for educational purposes and evaluated for effectiveness [3, 10]. To our knowledge, neither VHDL nor other hardware description languages were addressed in the scope of such visualization environments, so far. VISUAL-VHDL is a first step in this direction. Compared to related systems for visualizing digital circuits, such as JLS [5] and DLSim 3 [9], our system focuses on the synthesis process.

Besides its advantage in visualizing the synthesis, VISUAL-VHDL embeds a toolbar which enables editing circuits in a drag and drop mode. Schematics generated in this way may be used for preparing lecture notes or slides.

The remainder of the paper is structured as follows. Section 2 provides a brief introduction into VHDL and a review of Animal, which our platform is based on. Section 3 details VISUAL-VHDL. Section 4 concludes the paper with a summary and an outlook.

2.2 Animal

ANIMAL is a Java-based system for visualizing algorithms and data structures [6]. ANIMAL takes as input a special ASCII-based script denoted as ANIMALScript, which defines the animation content [7]. Each line in ANIMALScript can represent a command consisting of a keyword and a number of parameters. Listing 2 shows a section of an ANIMALScript file. Line 1 defines and displays a triangle. A hidden polyline is specified in line 2. Line 3 tells the triangle to smoothly move along this line within 3 seconds.

3. VISUAL-VHDL

VISUAL-VHDL extends both the graphical library of ANIMAL and ANIMALScript. Figure 1 shows the general flow for generating an appropriate animation for a given VHDL code. During its analysis, the VHDL code can optionally be optimized on the Boolean level based on the Quine-McCluskey...
The next task is to generate an extended netlist, which is a structural description of the digital circuit enhanced with visualization and animation information. This information is generated in the style of ANIMALSCRIPT, so that it can be treated by ANIMAL. The circuit primitives of the extended netlist are selected from a graphical library, which was extended with new classes to support digital logic schematic. Besides the automatic approach, VISUAL-VHDL allows the generation of an extended netlist from a schematic editor.

The core functionality of VISUAL-VHDL consists in interpreting the extended netlist and constructing an animation that can be viewed inside ANIMAL, utilizing the Java Swing library. This animation program can then be executed under user interaction to generate the circuit schematic corresponding to the analyzed VHDL code step by step. The user interaction is performed within the animation window with video player-like controls.

Figure 2 shows a screenshot of the animation window in an early step of the visualization process. Note how VISUAL-VHDL highlights the code row (s2 <= not c), which relates to the currently visualized digital inverter. In the final step, the animation window appears as shown in Figure 3.

VHDL Code
Code Analysis and Optimization
Generate Extended Netlist
Primitives
Effects
Extended Netlist
Generate Animations
Drag&Drop Schematic Editor
Figure 1: Generation of an Animation for VHDL Models

Algorithm [11]. If desired, this optimization process can also be visualized step-by-step.

The extended graphical library is based on the graphical library of ANIMAL and supplements it with new primitives to display logical gates, flip-flops, multiplexers, entities etc. The new special class Wire in VISUAL-VHDL is used to connect the terminals of different primitives.

In the following, we describe some important aspects of VISUAL-VHDL in more detail.

3.1 Code Analysis and Optimization

In its current prototype, VISUAL-VHDL supports the following VHDL language constructs: entities, ports, signals, architectures, processes, concurrent and sequential signal assignments, variables and variable assignments, conditional assignments, component declarations and instantiations.

From a design perspective, VISUAL-VHDL allows the visualization of VHDL models which presents behavioral, structural, and mixed models, i.e., models with both behavioral statements and component instantiation. For behavioral models, both combinational and sequential logic are supported, so that a description on the register transfer level (RTL) is allowed. Description on the RTL level is the most well-known approach to specify hardware in commercial design flows. On this level, the designer takes the responsibility for bit-accurate resource allocation, task mapping, and cycle-accurate scheduling. Thus, with the aid of VISUAL-VHDL, students do not only learn VHDL and digital logic, but the de facto standard design approach on the RTL level.

Upon parsing and analyzing the VHDL model, VISUAL-VHDL offers an optimization of this model. VISUAL-VHDL currently uses the Quine-McCluskey algorithm [11] to find the minimal form of the Boolean function represented by the VHDL model. Students can switch the optimization process on and off to learn the effect of optimization on the synthesis result. Figure 4, for instance, visualizes the synthesis result of a given VHDL code without optimization. Investigating the code shows that the signal s2 can simply be determined by inverting the signal c as c.d + cd = c. Thus, if the same code is visualized with optimization, the synthesis result would appear as given in Figure 3.

Figure 2: Example VHDL Animation, Currently Processing the Assignment of Signal S2

The core functionality of VISUAL-VHDL consists in interpreting the extended netlist and constructing an animation that can be viewed inside ANIMAL, utilizing the Java Swing library. This animation program can then be executed under user interaction to generate the circuit schematic corresponding to the analyzed VHDL code step by step. The user interaction is performed within the animation window with video player-like controls.

Figure 2 shows a screenshot of the animation window in an early step of the visualization process. Note how VISUAL-VHDL highlights the code row (s2 <= not c), which relates to the currently visualized digital inverter. In the final step, the animation window appears as shown in Figure 3.

Figure 3: Example for VHDL Animation (Last Animation Step)
is taught in many courses on logic design. Besides its usage during synthesizing the VHDL code, VISUAL-VHDL provides the possibility to visualize the proceeding of this algorithm. By this means, students learn about processes running in the background of the synthesis task.

Figure 4: Synthesis Result Without Optimization

3.2 Extended netlist

The extended netlist includes all the information needed for the dynamic visualization of the digital circuit. An extended netlist extends ANIMALSCRIPT with several primitives to visualize circuit symbols. During the animation process, an extended netlist is processed sequentially from top to bottom. Listing 3 shows the section of the extended netlist responsible for visualizing the flip-flop and its connection with the OR-gate according to Figure 3. In particular, this script section contains two animation steps parenthesized with curly braces {}:

1. The first step performs the following four actions simultaneously. (1) The fillcolor of the OR-gate is removed. (2) A D-flipflop is visualized, with upper-left and lower-right corners at the position (421,117) and (571,267), respectively. Note that the y-axis in VISUAL-VHDL is directed downwards. The names of the inputs and outputs of the flip-flop are specified. The attribute color specifies the color of the flip-flop frame and the signal names. The flip-flop is finally highlighted by a fillcolor. (3) Line 16 in the VHDL code, which was highlighted in the previous animation step, is unhighlighted. (4) Line 17 (c0 <= s3) is highlighted.

2. In the second step, a wire from the output of the OR-gate to the input of the D-flipflop is visualized. Note that this wire is specified by three points, as schematically depicted in Figure 5. See Figure 3 for comparison.

<table>
<thead>
<tr>
<th>Listing 3: An Extended Netlist Section Relating to Figure 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {color &quot;or1&quot; type &quot;fillColor&quot; none</td>
</tr>
<tr>
<td>2 d &quot;d1&quot; (421,117) (571,267) input &quot;s3&quot;</td>
</tr>
<tr>
<td>output &quot;co&quot; clock &quot;clk&quot; reset &quot;rst&quot;</td>
</tr>
<tr>
<td>color black fillColor (153,153,255)</td>
</tr>
<tr>
<td>3 unhiglightCode on &quot;codeSource&quot; line 16</td>
</tr>
<tr>
<td>4 highlightCode on &quot;codeSource&quot; line 17</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6 wire wire,or1−0&gt;d1−0 (404,192) (404,178)</td>
</tr>
</tbody>
</table>

Figure 5: Illustrating the Wire Dimensioning

The automatic generation of an extended netlist for a given VHDL code is a highly complex task, which includes the following subtasks:

1. Determining the optimal placement of the logical elements.
2. Determining the optimal routing.
3. Determining the optimal animation.

The execution of the first two subtasks results among others in the (x, y) position data for all the elements and wires of the circuit. These data are supplied as parameters in the extended netlist, see Listing 3.

As mentioned in Section 3, an extended netlist is generated at the end of the analysis and optimization phase. For this purpose, each element is specified by an attribute, which gives the number of the VHDL code line relating to that element. This attribute is required during the construction of the extended netlist to highlight the VHDL lines and the corresponding circuit elements synchronously.

For an appropriate display of the digital circuit, the animation field of the animation window is organized as a grid of numbered cells and channels. Logical elements are placed into cells, while wires are laid within the channels.

The grid size in terms of cell number and size is automatically adjusted according to the netlist content. Some rules are defined for simplifying the placement process. One rule relates to the placement of the circuit elements having external interface. Elements with external input signals, for instance, are placed in the leftmost grid column as far as possible. In contrast, elements with external output signals are placed in the rightmost grid column.

Visualizing wires in VISUAL-VHDL is a sophisticated task for the following reasons:

1. Wires should be as short as possible.
2. Wire segments can be only horizontal or vertical, not diagonal.
3. Suitable inflection points should be found.
4. Intersections should be minimal.
5. Wires should expand from the output of an element to the input of another. The expansion velocity should be adjustable.
To provide this flexibility, a Wire object is realized as a set of points. To visualize a wire consisting of several segments, the start points of each segment and the end point of the last segment are provided as parameters, see Figure 5.

3.3 Schematic Editor

The schematic editor as shown in Figure 6 is an extension of the graphic editor of ANIMAL. The new digital toolbar at the right includes symbols for 15 element types including gates, flip-flops, multiplexers and demultiplexers. Upon dragging and dropping a symbol, several parameters can be set, such as the name and the color. The number and the names of inputs can be entered for each gate. For a multiplexer, for instance, the user may set the number of the data inputs. The number of the control signals is then determined internally to avoid errors. A D-flipflop can optionally be provided with set, reset and/or clock-enable signals.

![Figure 6: VISUAL-VHDL Schematic Editor](image)

```
In addition to plotting, the schematic editor enables the simulation of simple combinatorial circuits. For this purpose, students can define the digital value for each input of the plotted elements. The simulation core determines the output values and visualizes them automatically.
```

4. CONCLUSION

VISUAL-VHDL is a visualization platform for VHDL. It enables entering VHDL code and an interactive production of circuit schematic. By this means, students can learn the effect of the versatile language constructs on the resource allocation, task mapping and scheduling in the target system. Educators may also take advantage of this tool to verify their models or to quickly generate circuit schematics using the drag and drop toolbar of the schematic editor. VISUAL-VHDL is the first step toward a sophisticated system to support the learning process in many subjects of computer and electrical engineering. We are currently developing a web interface for VISUAL-VHDL. Besides facilitating the usage of our tools, the web interface includes a feedback system enabling students to evaluate these tools, so that we can prove their effectiveness in the near future. Furthermore, our platform will be completed and developed to support further features of VHDL and its event-driven simulation process. Other hardware description languages, such as Verilog, will also be considered.

Learning algorithms for computer aided design is important, especially for computer science students. Algorithms for synthesis, optimization, placement, and routing, which usually run in the background of commercial tools, will come to the fore by VISUAL-VHDL. By these means, students will not only learn how these algorithms work. They also will become acquainted with an interesting field for their application.

5. REFERENCES


