International Journal of Engineering Education Vol. 35, No. 4, pp. 1224–1237, 2019
Printed in Great Britain

0949-149X/91 $3.00+0.00

A Web-Based Tool for Learning Digital Circuit High-Level Modeling*

ANDREJ TROST and ANDREJ ŽEMVA
Faculty of Electrical Engineering, University of Ljubljana, Slovenia. E-mail: Andrej.Trost@fe.uni-lj.si

Modeling of increasingly complex digital circuits requires high-level design tools and languages. Design tools based on standard hardware description languages need substantial training which limits their usage in a typical digital system design course. In this paper, we propose a small hardware description language SHDL for circuit modeling and present a novel web-based educational tool. The SHDL includes similar high-level modeling semantics to VHDL but has substantially simplified syntax. The web-based SHDL modeling and simulation tool enables quick introduction to language-based circuit modeling. The SHDL model is automatically translated to the equivalent VHDL model. The proposed tool enables conducting more laboratory and self-learning circuit design experiments important for comprehending the design process. A one-semester laboratory hardware modeling lessons are described and educational experience with the new methodology is discussed.

Keywords: teaching digital design; high-level model; hardware description language; educational web tool

1. Introduction

Digital design is one of the key topics in electrical and computer engineering education. A fundamental digital design course covers switching circuits, logic gates, Boolean equations, combinational and sequential circuit design and analysis. The curriculum is traditionally based on switching circuits and logic chips. Due to the advances in digital integrated circuits technology, programmable logic devices and hardware description languages (HDL) are often included in the curriculum. The digital design teaching books include HDL modeling in one of the standard languages: VHDL [1] or Verilog [2]. A recent survey [3] shows that a little more than half of the electrical and computer engineering digital courses are HDL based. The digital design topics are spread over the study programs in courses from the first year of study to advanced courses for master students. A digital design course should be focused on the design principles and not on the details of the HDL syntax and development tools. Teaching and learning the high-level circuit modeling can benefit from a tool that supports building the HDL model from a simplified description and evaluating the design in a web browser.

At our Faculty of Electrical Engineering, we begin with a basic digital structures course for sophomore and continue in the next year to advanced digital integrated circuits and systems. The lectures are focused on digital integrated circuit analysis and design of digital systems. The laboratory practice cover HDL design with programmable devices. Hands-on experiments are important for learning digital design. Traditional experiments with TTL logic chips on breadboards are today superseded by logic simulators and programmable development boards. The subject teachers can choose among a variety of commercial and free logic simulators to support supervised practical work and self-learning [4].

Programmable devices enable conducting a broad range of experiments, from basic logic circuits with simple input/output interfaces to systems with digital controllers and standard interfaces. The students begin to learn design techniques by examples of standard digital building blocks and advance by implementing unique projects on FPGA boards [5–10]. Projects using VGA graphics port to display data on a computer monitor are among most popular because the VGA controller is easy to implement on the FPGA and the students are motivated to design simple games or animation [5–7]. Other common educational FPGA design project topics are application specific signal and image processing circuits [8–9] and custom microprocessor design [10].

Advanced digital systems courses introduce system-on-chip devices with programmable logic and microprocessor on a single programmable integrated circuit [11]. Programmable devices require a development environment, referred as back-end tools, provided by the device manufacturer vendors. The back-end-tools contain HDL synthesis and technology implementation compilers and advanced options for efficient digital systems development beyond HDL: block design, intellectual property components and high-level synthesis [12]. These tools are developed for trained engineers and are not well suited for the digital design introduc-

* Accepted 24 April 2019.

tion. From our experience, the tool installation and usage complexity discourage the students to set-up the tool for self-learning. Several web-based educational tools were developed to support the teaching of digital logic. System for Digital Logic Design and Simulation [13] allows users to design and simulate switching circuits. A remote laboratory setup can be used for circuit design and testing [14, 15]. A lightweight web environment enables fast development and testing digital designs [16]. Digital simulator is the main analysis tool for the digital designer and visualization of simulations supports digital design education [17].

Students learning digital design with HDL have difficulties to comprehend the unfamiliar language syntax and modeling paradigm [18]. The main steps of the HDL based digital design and prototyping process are:

(1) design specification,
(2) building the digital circuit model,
(3) verification and prototyping implementation.

The specification describes the task of the digital circuit and is usually in the natural language form, which needs to be transformed into the digital circuit model. The digital model is described on the register-transfer level (RTL) using hardware description language. The transformation requires knowledge of the modeling syntax and semantics as well as understanding the technology implementation. Digital circuit model is different to a computer program in several ways; the logic expressions are executed in parallel, the timing is specified in terms of clock cycles and there are many technological modeling limitations.

The industry standard language VHDL has verbose syntax and the students are required to learn a lot of keywords and semantic rules even for small digital circuit models. The semantic rules define limitations for signal assignments order in concurrent or sequential statements. Designing a synthesizable circuit description requires careful modeling of synchronous and asynchronous logic.

Several tools and languages on the higher abstraction level were developed to mitigate the digital design complexity and close the gap between design efficiency and technology resources [19]. These tools translate models to lower level description usable for back-end synthesis and technology mapping. To efficiently use the abstract tools, the designer still needs the knowledge of basic logic elements and practical understanding of the RTL languages such as VHDL. Several educational languages and supporting tools were proposed to introduce HDL based digital design [20, 21], but none of these languages can be used to guide students to adopt proper VHDL coding practice. Knowing industry standard HDL coding should be one of the digital course outcomes.

We propose a small hardware description language SHDL and we developed a web-based educational tool to boost learning language based high-level digital circuit design. The web tool enables SHDL modeling, simulation and transformation to functionally equivalent VHDL code. In the next section, the HDL modeling, SHDL syntax and the supporting web tool are presented. Introduction and usage of the proposed methodology in digital systems design laboratory is discussed in Section 3. Section 4 concludes the paper with the summary of the experience and guides for the future.

2. Hardware description languages in digital systems education

Electrical engineering digital design education is composed of several courses covering topics from digital integrated circuits structures and logic building blocks to digital systems. The initial digital design course follows a bottom-up approach starting with thorough explanation and analysis of simple logic structures. Smaller logic circuits can be adequately described schematically, but a hardware description language is also introduced to set a firm ground for building larger digital circuits and systems.

A higher level of abstraction provided by the hardware description languages is required for the efficient design of contemporary digital systems. High-level hardware description languages use three modeling styles: structural, data-flow and behavioral. The structural modeling is used for hierarchical design, where the circuit is composed of smaller interconnected building blocks. The structural model is equivalent to low-level digital schematic design. The development environment tools allow heterogenous description of a digital system, which is typically composed of a schematic for the top level and HDL description for the operation of building blocks.

Data-flow modeling describes the operation of the digital system where input data is transformed to produce output signals. Transformations are described using concurrent modeling style where a series of statements are operating in parallel. The benefit of a high-level language in data-flow models is a rich set of available data types and operators. For example, when we use an adder in schematic design we must select a specific library component matching input data type and size. The same adder can be described in HDL with a simple expression.

Behavioral modeling is the highest abstraction level in a hardware description language. The model describes the circuit operation or algorithm and not

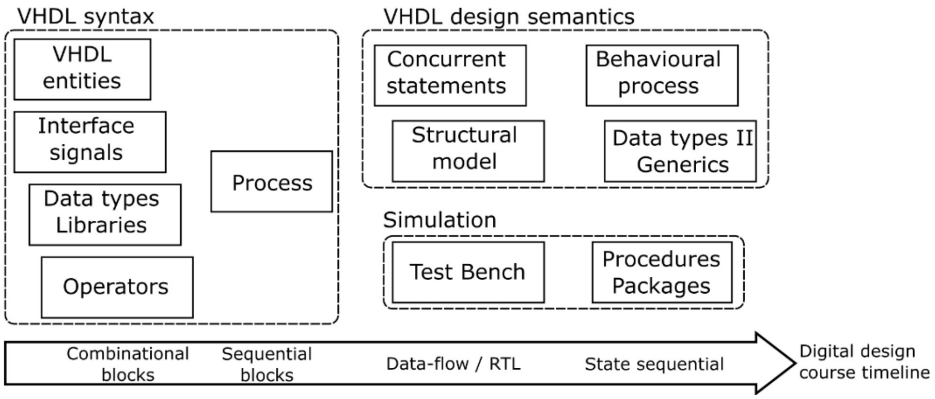

Fig. 1. VHDL teaching topics covering bottom-up digital design course.

explicitly the circuit structure or flow of data. Behavioral models introduce language constructs like software languages and a concept of a statement sequence for describing the algorithm.

The language VHDL used in our digital design courses has a very verbose syntax and strict typing rules causing most of the learning problems to the students. Fig. 1 presents the VHDL topics following the bottom-up digital design course. The language structures: entities, interfaces, data types, operators and process must be explained to model basic combinational or sequential blocks. Even small logic building block models in VHDL contain a dozen keywords. Register-transfer level data-flow circuits introduce concurrent statements and structural modeling concepts: VHDL components and port maps. Structural and discrete event modeling is used to build simulation test benches. Design of a state-based sequential circuit, e.g., counter or finite-state machine is based on a VHDL behavioral process. Larger logic models benefit from advanced data types, generic parameters and packages with procedures used for advanced simulation.

2.1 VHDL topics covering bottom-up digital design

A course on digital systems aims to upgrade the knowledge based on a top-down digital system design approach. The students should be able to develop a model of a digital system according to specifications and present a working design project. The practical learning should be focused on the modeling semantics, but we experience that a lot of time is wasted for writing syntactically correct VHDL code. Paying a lot of attention to the syntax takes time which could be used to improve understanding of semantics and reasoning behind the construction of digital system models.

To augment VHDL design efficiency, we propose a small hardware description language—SHDL which has simple syntax rules. Fig. 2 presents the SHDL topics covering bottom-up digital design course. The SHDL model of combinational blocks requires only combinational assignments with operators and sequential blocks introducing sequential assignments. To build a simple test bench we propose a graphical test bench design tool incorporated in the simulator. Concurrent statements description rules are used for data-flow models and conditional statement for behavioral modeling of the state sequential circuits. The SHDL should be automatically translated to VHDL for the back-end implementation tools.

2.2 Small hardware description language

The modeling language SHDL is constructed based on our experience with hardware description languages VHDL and Verilog for digital systems education. Educational usage is typically constrained to smaller projects requiring only a limited set of the language syntax. The SHDL is not intended to model every aspect of the digital systems

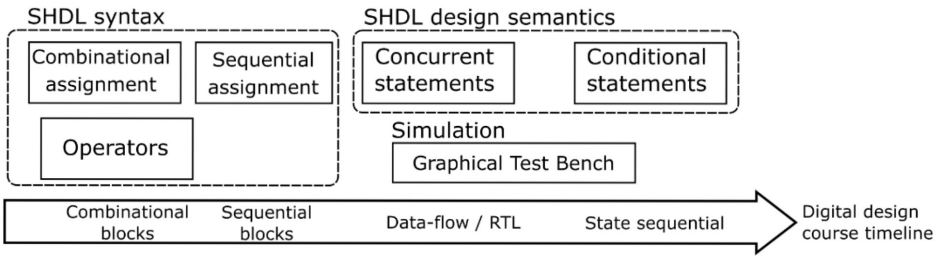

Fig. 2. Proposed SHDL topics in bottom-up digital design course.

```
<expression> :== <bool> [ 'or' | 'xor' <bool> ]
<bool> : == <relation> [ 'and' <relation>]
<relation> :== <shift> [ <relational_operator> <shift> ]
<relational_operator> :== '=' | '/=' | '<' | '<=' | '>' | '>='
<shift> :== <simple_expression> [ '<<' | '>>' <number> ]
<simple_expression> :== <term> [ '+' | '-' | '&' <term> ]
<term> :== <factor> [ '*' <factor> ]
<factor> :== <primary> | '-' <primary> | 'not' <primary>
<primary> :== <name> | <slice> | <number> | '(' <expression> ')'
<slice> :== <name> '(' <index> ')'
<index> :== <name> | <range>
<range> :== <number> [ ':' <number> ]
```

Fig. 3. Backus-Naur form syntax of expressions in input high-level language using VHDL mode.

and is intentionally limited to the most commonly used behavioral hardware description structures used in digital systems education. The students should get a firm understanding of the basic language syntax and modeling semantic rules first.

The core of SHDL is composed of the most useful operators which can be expressed with VHDL syntax or ANSI C syntax, for the designers more comfortable with a software language. The operators include:

- Boolean logic operators: and, or, xor, not in VHDL (&, |, ^, ~ in C),
- arithmetic operators: +, – (including unary), * in VHDL and C and
- relational operators: =, /=, >, >=, <, <= in VHDL (==, !=, >, >=, <, <= in C).

The formal rules for building expression described in a Backus-Naur form (BNF) are presented in Fig. 3. The rules define expression parsing algorithm and the priority of the operators.

In VHDL mode, we include some hardware design language specific operators connected with vector data type: concatenation (&) and slicing (:). These operators model combining or dividing multiwire bus with zero logic resource usage, which is important for efficient hardware design.

There are a lot of data types in the VHDL syntax and libraries, but in practice, only a limited set is used for building synthesizable circuit models. The VHDL syntax rules require explicit conversion for assignment of different data type and smaller set of data types mitigate several conversions. Basic data types represent one-bit signals, multi-bit vectors, signed and unsigned vectors and integers. The circuit models mainly use vector data types since they most efficiently represent logic resources. Integer signal type has limited usage in VHDL because it is synthesized to a 32-bit bus if not specifically constrained and cannot be concatenated or sliced.

Table 1 presents basic signal data types in VHDL and corresponding simplified types in SHDL. One-bit signal declaration in SHDL is either empty (default) or u1, which is equivalent to one-bit unsigned vector. The vectors are declared as signed (uN) or unsigned (sN), where N is the vector size. An equivalent of the integer in VHDL is s32.

Literal values in VHDL have a variety of notations. One-bit values require single quotations, vector values are binary strings within double quotations and decimal integers are represented without quotation marks. Assigning a decimal number to a vector is not possible without radix specifier or conversion function, for example: `d <= "0000";` can be expressed as: `d <= to_unsigned(0, 4);` and not simply: `d <= 0`. We simplify these rules in the SHDL, where an integer value in decimal, binary or hexadecimal format can be assigned to signal of any data type. When the

Table 1. Basic signal data types in VHDL and corresponding SHDL

VHDL		SHDL		
type declaration	**example value**	**declaration**	**example value**	**binary value**
`std_logic`	`'0','1'`	`empty` or `u1`	`0, 1`	`0b0, 0b1`
`std_logic_vector (N-1 downto 0)`	`"0010"`	`uN`	`2, 0b0010`	`0b0010`
`signed (N-1 downto 0)`	`"1110"`	`sN`	`-2`	`0b1110`
`unsigned (N-1 downto 0)`	`"1010"`	`uN`	`10, 0b1010`	`0b1010`
`integer`	`5`	`s32`	`5`	`0b101`

code is translated to VHDL, the corresponding binary string notation or conversion function is used.

The register-transfer level HDL models describe synchronous digital circuits using expressions for transfer functions and flip-flop or register models. Components of a hierarchical digital system are typically single clock synchronous circuits. The logic circuits designed for programmable devices contain one or only a few clock signals due to the limitation of on-chip clock routing resources. Circuits with more than one clock signal should contain synchronization logic, are difficult to simulate and analyze. We are thus avoiding circuits with multiple clocks for the introductory course of digital design.

Synchronous circuits update registers either on rising or on the falling edge of the clock signal. Only some specific interface circuits require both types of registers while most sequential components use only rising edge registers. Models of sequential circuits in SHDL are constrained to a single clock signal which enables further simplification of the language syntax. We define two assignment operators: combinational assignment is declared with operator '=' and synchronous sequential assignment with '<=', as described in BNF:

```
<assignment_statement> :== <target_
  name> '=' | '<=' <expression>
```

The combinational assignment is used to model logic which is driving the target signal. The synchronous sequential assignment adds a register to the logic and the target signal gets updated on the rising edge of the clock. The assignment operator syntax is borrowed from the language Verilog, where non-blocking assignment '<=' is used for sequential building blocks and continuous assignment '=' for combinational.

Behavioral models in HDL have a concept of sequential statement blocks, e.g., process in VHDL where the order of statements is important in contrast to concurrent statements in the data-flow model. In a sequence of assignment statements with the same target signal, only the last assignment will be executed. Such assignments should be inside conditional statements to produce useful code. The conditional statement syntax is presented in Fig. 4.

2.3 Web tool for SHDL

A web tool was developed to support modeling with small hardware description language. The web scripting language JavaScript is powerful enough to build a complete design environment targeting educational usage. The advantage of the web-based tool is that it requires no specific tool installation, its availability in different platforms regardless of size and operating system. We have previous good experience in specific web tools for digital circuit education; we already designed Graphical Test Bench, Micro-operation design tool and educational central processing unit compiler.

The web tool is based on JavaScript (ES6), HTML 5 and W3CSS. The tool is freely available online [22]. The JavaScript code [23] is divided into 7 modules:

- Userint.js—user interface functions, connections with HTML,
- Vector.js—define 64 bit signed/unsigned vector and operations,
- Model.js—methods for high-level circuit model,
- Lexer.js—code lexical analysis,
- Parsesim.js—code parser and discrete event simulator,
- Wave.js—simulation waveform methods and
- Vhdl.js—code translation to VHDL.

A screenshot of the web tool is presented in Fig. 5. The tool has an easy to use interface where the main design steps are connected with buttons: Parse—parsing the SHDL code, Run—running simulation, Ports – opening signal table, VHDL—translating SHDL to VHDL and TestBench—generating VHDL test bench code.

The work area is divided into code window, ports and signals table, parser messages window and waveform canvas. The designer can graphically set input values in the waveform and observe results after running simulation.

Fig. 6 presents the main tasks and data in the SHDL modeling design flow. The parser inputs are SHDL code and data from the signal table. If parsing is successful, an internal circuit model is produced along with a resource utilization report. The report summarizes circuit components that were identified during code parsing: number of input/output pins and flip-flops, number of logic

```
<if statement> : == 'if' '(' <condition> ')' <statement> | '{' <statement block> '}'
                 [ 'else' <statement> | '{' <statement block> '}' ]

<statement block> :== <statement> [';'] { <statement> [';'] }
```

Fig. 4. BNF syntax of conditional statement and statement block in SHDL.

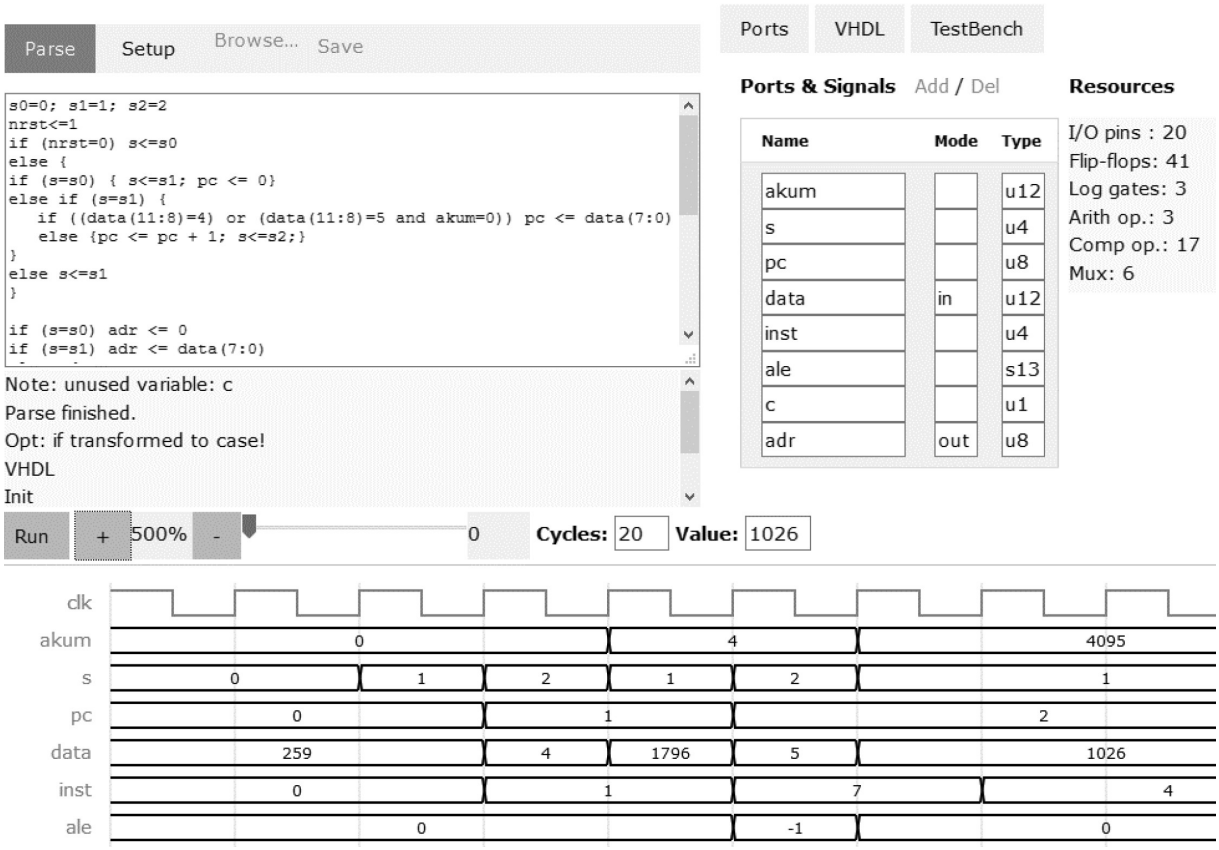

Fig. 5. Web tool user interface with SHDL code editor and parser messages on the left, ports & signals table and resource report on the right, and simulation waveform at the bottom.

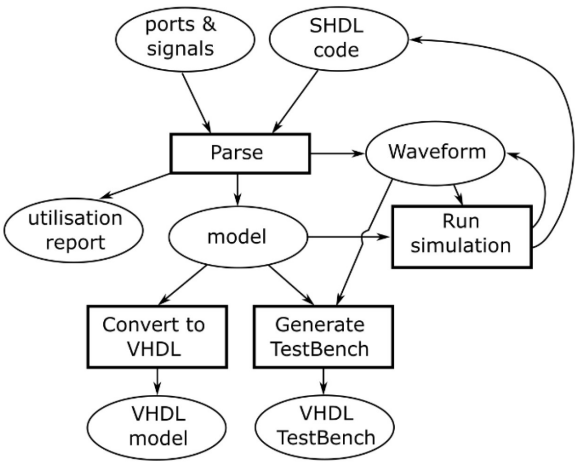

Fig. 6. SHDL modeling design flow tasks (bold rectangles) connected with the web tool data components (ovals).

gates, arithmetic and comparison blocks extracted from operators and number of multiplexers. This is like the VHDL tools synthesis report, but on a higher abstraction level, since the tool lacks information about the target technology. It can be used by the designer as a guideline for the complexity of the produced circuit and comparison of different coding approaches.

Waveform signals are displayed with their default (zero) values after initial parsing the code. To prepare the simulation, the designer should set the number of clock cycles and input values. The tool supports intuitive graphical setting of the input values on the waveform canvas. Running the simulation evaluates the model for all cycles and presents output and internal signal values on the waveform. The circuit designer's task is to check the responses of the circuit to the input stimuli. The tool supports zooming and panning the waveform and three data display modes for vector signals: integer, binary and analog. According to the results, the designer decides to convert the model to VHDL code or upgrade the SHDL and rerun the parsing and simulation. The tool also produces the test bench which can be used for VHDL simulation with the same waveform settings.

3. SHDL in the digital systems design laboratory

Our students enrolled in digital systems design already learned the basics of VHDL circuit modeling. In the laboratory practice, they design some educational digital building blocks, develop a

custom microprocessor and finally a project on their own.

3.1 Introduction tutorial

We consider a 2–4 binary decoder circuit design to introduce the SHDL logic level and behavioral modeling concepts and the usage of the SHDL web tool.

Fig. 7 presents a declaration of one 2-bit unsigned input signal (a) and four one-bit output signals (y0, y1, y2 and y3). The digital circuit can be described with a logic diagram or with hardware description language using logic expressions, as presented in Fig. 8. We describe the model in the web-based tool, parse the model and check the operation with the integrated logic simulator. For the simulation, we manually set the input values and observe the outputs on the waveform as presented in Fig. 8c.

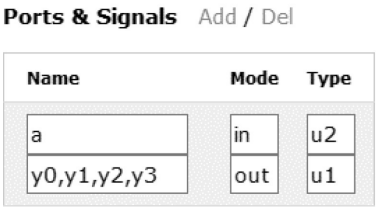

Fig. 7. Example of port and signal declaration for 2–4 binary decoder.

This is a low-level logic model of a circuit based on Boolean equations. The model can be automatically translated to VHDL model and the VHDL test bench as presented in Fig. 9.

The assignment statements with logic expressions in VHDL are almost the same as in our model with an exception of the SHDL combinational assignment operator. The most notable difference is in the

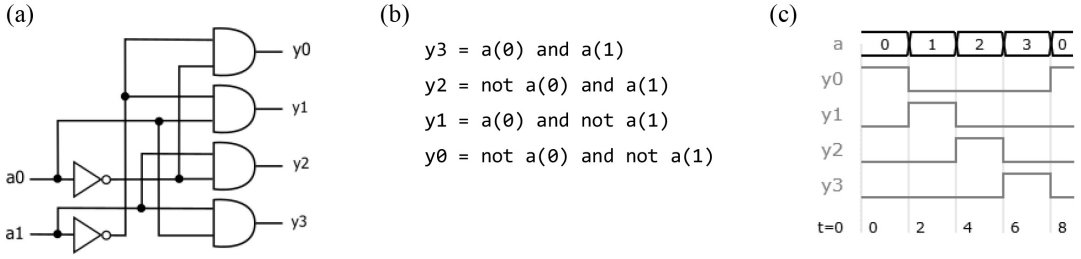

Fig. 8. (a) Logic diagram, (b) SHDL logic expressions and (c) simulation of a 2–4 binary decoder.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Mux is
 port (
   a : in unsigned(1 downto 0);
   y0,y1,y2,y3 : out std_logic
 );
end Mux;

architecture RTL of Mux is
begin
y3 <= a(0) and a(1);
y2 <= not a(0) and a(1);
y1 <= a(0) and not a(1);
y0 <= not a(0) and not a(1);

end RTL;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Mux_tb is
end Mux_tb;

architecture sim of Mux_tb is
 signal a : unsigned(1 downto 0);
 signal y0,y1,y2,y3 : std_logic;
 constant T : time := 10 ns;
begin

uut: entity work.Mux port map(
     a => a,
     y0 => y0,
     y1 => y1,
     y2 => y2,
     y3 => y3
);

stim_proc: process
begin
 a <= "00";
 wait for 2*T;
 a <= "01";
 wait for 2*T;
 a <= "10";
 wait for 2*T;
 a <= "11";
 wait for 2*T;
 a <= "00";
 wait;
end process;
end sim;
```

Fig. 9. Output VHDL code and test bench for 2-4 binary decoder.

VHDL language verbosity and structure, where VHDL requires libraries and specific modeling constructs (entity, port, architecture).

The port signals are transformed to internal signals in the VHDL test bench and the circuit instance is defined using a port map. The test bench generator parses waveform and outputs a process with the sequence of assignments to the input ports and wait statements with the corresponding cycle delay.

The test bench can be used to verify the generated VHDL model in the external VHDL simulator which should produce identical results. The designer can also use declaration and port map statements from the test bench to include the generated VHDL model as a component in structural design.

A high-level model of the same circuit defines the behaviour of the binary decoder: if input combination is 0, the output y0 is active and others are 0, if a combination is 1, the output y1 is active, etc. The circuit can be described in SHDL using four if statements:

```
if (a=0) {y0=1; y1=0; y2=0; y3=0}
if (a=1) {y0=0; y1=1; y2=0; y3=0}
if (a=2) {y0=0; y1=0; y2=1; y3=0}
if (a=3) {y0=0; y1=0; y2=0; y3=1}
```

The modeling language is using the concise syntax of if statements like C-language which is translated to the verbose syntax of VHDL. For example, the first statement translates to:

```
if a = 0 then y0 <= '1'; y1 <= '0'; y2 <= '0';
y3 <= '0'; end if;
```

We can further improve the readability of the model: instead of setting all outputs in every if statement, we define default values at the beginning and describe only change of the outputs. Since conditions refer to the same input signal, we choose an if-else structure and omit the last condition. Fig.10 presents the model and an excerpt of the generated VHDL code with a combinational process. The VHDL converter detects a sequence of conditional statements where the same variable is compared to different values and translates the sequence to a case statement in the output code.

A sequence of conditional statements is also used to model a priority encoder, where the order of comparison is important. The circuit model and generated VHDL combinational process are presented in Fig. 11. The VHDL process includes if-elsif structure and data type conversion functions are used to assign integer values to 2-bit unsigned output.

A multiplexer circuit with enable input signal is an example of a combinational circuit used to demonstrate semantic rules in behavioral modeling. Fig. 12 presents three models of a 4–1 multiplexer in SHDL. The first model contains a sequence of conditions describing multiplexer inside enable condition to produce regular output or zero if enable is not active. Combinational circuit model must define output in all cases and to obey this semantic rule a complete form of conditional statement is used with assignments to output in if and else blocks. If the outputs are not defined in all cases, the HDL code

```
y0=0; y1=0; y2=0; y3=0
if (a=0) y0=1;
else if (a=1) y1=1
else if (a=2) y2=1
else y3=1
```

```
process(all)
begin
  y0 <= '0'; y1 <= '0'; y2 <= '0'; y3 <= '0';
  case a is
   when "00" => y0 <= '1';
   when "01" => y1 <= '1';
   when "10" => y2 <= '1';
   when others => y3 <= '1';
  end case;
end process;
```

Fig. 10. SHDL model of 2-4 binary decoder and translation to VHDL combinational process.

```
if (d3=1) y=3
else if (d2=1) y=2
else if (d1=1) y=1
else y=0
```

```
process(all)
begin
  if d3 = '1' then
     y <= to_unsigned(3,2);
  elsif d2 = '1' then
     y <= to_unsigned(2,2);
  elsif d1 = '1' then
     y <= to_unsigned(1,2);
  else
     y <= to_unsigned(0,2);
  end if;
end process;
```

Fig. 11. SHDL model of priority encoder and translation to VHDL.

```
(a) if (en) {
      if (s=0) o = d0
      else if (s=1) o = d1
      else if (s=2) o = d2
      else o = d3
    } else o = 0
```

```
(b) if (s=0) o = d0
      else if (s=1) o = d1
      else if (s=2) o = d2
      else o = d3
      if (en=0) o = 0
```

```
(c) o = 0
      if (en and s=0) o = d0
      else if (en and s=1) o = d1
      else if (en and s=2) o = d2
      else if (en) o = d3
```

Fig. 12. Three behavioural SHDL models of 4-1 multiplexer with enable input.

synthesis tool generates a latch to hold the last defined value and the synthesized circuit is not combinational.

A model in Fig. 12b introduces negated enable condition after the sequence of conditions describing the multiplexer. The sequence connects the output to one of the inputs, but if the enable signal is not active (en=0), the output is 0. The order of these conditional statements is important in SHDL as well as in VHDL. An inexperienced designer would try to write enable condition at the beginning producing failure in the model.

The third behavioral model presented in Fig. 12c defines a default value of the output before a sequence of conditional statements. This is a common good practice of describing combinational circuits with a set of not complete conditions. The default value ensures that the combinational semantic rule is fulfilled.

Sequential building blocks: registers and flip-flops are described in SHDL with a simple sequential assignment:

```
q <= d
```

A binary counter is defined as:

```
cnt <= cnt + 1
```

and a corresponding data type declaration in ports and signals table. By adding conditional statements, students can quickly create models of registers and counters with reset and enable control signals, modulo counters etc. These circuits are already part of the laboratory design exercises.

3.2 Laboratory circuit design exercises

Table 2 presents a selection of laboratory exercises for a one-semester course on digital systems. The initial exercises are typical educational examples: data comparison and sorting circuit, serial registers and counters. The student's task is to develop a circuit model according to specifications, verify the model with the simulator, synthesize the circuit from the VHDL description and test on the FPGA development board. The students can either develop models directly in VHDL or use SHDL and the web-tool for the first design steps.

Columns in Table 2 report amount of lines (LI) in SHDL and the corresponding amount of lines and keywords (KEY) in VHDL. The SHDL model has on average 4-times fewer lines of code compared to the VHDL model. The high-level circuit model characteristics are represented by the number of input/output signals (IO), flip-flops (FF), operators (OP) and multiplexers (MUX).

The next three laboratory exercises are examples of data-flow synchronous sequential circuits. Pulse-with modulator (PWM) is a useful digital system building block. A basic model of the 3-bit PWM circuit generating pulses of different width on the output is presented in Fig. 13. The pulse widths are defined by the input vector w compared to the modulo-6 counter producing output waveform presented in Fig. 13c. The SHDL model is considerably smaller than VHDL enabling students to spend more time experimenting with the circuit operation. For the laboratory exercise, they design advanced PWM and test the circuit using switches and LEDs on the development board.

Numerically controlled oscillator (NCO) is a digital oscillator generating periodic signals with variable frequency. The NCO circuit uses the overflow of an accumulator for the phase signal and a look-up table to produce output signal samples. It is a basic building block of a digital function generator or a source of data for signal processing circuits, for example, digital filters.

A finite-response filter (FIR) structure has a series of registers for delay line and multiply-add operators to compute the response. Filter coefficients are real numbers which require a complex circuit for the operators, but in many cases, we can use approximate computation with N-bit integers. If input and output data are integer numbers, we can scale the coefficients with 2^N, compute the response using rounded integer values and apply integer 2^N division. A small 4-tap FIR can be expressed in SHDL in only five lines of code:

```
x1 <= x0
x2 <= x1
x3 <= x2
sum <= x0*157+x1*355+x2*355+x3*157
res = sum(17:10)
```

Fig. 14 presents the online simulation of the FIR using a sinus sweep generator and analog display of input and output waveform.

The last two sets of the laboratory exercises are a VGA graphics controller and a small processor used to build digital system for the student's final projects.

The video graphics controller is composed of a VGA timing generator circuit, a sprite display and a coordinate transformation logic. The circuit components are separately designed and verified on the simulator. The final circuit is described in VHDL using structural modeling and tested on the FPGA board connected to a computer monitor. The stu-

Table 2. A selection of laboratory circuit design exercises with SHDL and VHDL model features: LI—code lines, IO—input/output pins, FF—flip-flops, OP—sum of high-level operators, MUX—multiplexers, KEY—number of different VHDL keywords and reserved words in the model.

Exercise	SHDL model					VHDL model	
	LI	IO	FF	OP	MUX	LI	KEY
Magnitude comparators							
(a) Comparator with flags	6	6	0	3	2	18	22
(b) 3-input sorting circuit	16	24	0	3	6	48	23
Registers							
(a) Serial in serial out shift register	10	3	4	1	4	38	22
(b) Linear feedback shift register	10	5	4	2	4	37	23
Counters							
(a) binary enable	2	6	4	3	1	29	26
(b) modulo counter	9	7	4	4	2	45	26
Pulse width modulator	8	4	7	3	2	34	25
Numerically controlled oscillator							
(a) Phase accumulator	3	5	10	1	0	26	23
(b) Oscillator with sinus table	23	8	10	18	2	70	28
Digital filter: 4-tap FIR	5	17	42	7	0	32	23
VGA							
(a) timing generator	11	26	24	10	4	51	25
(b) sprite	8	60	1	5	1	36	24
(c) coordinate transformation	6	14	0	5	2	39	22
Small CPU							
(a) control logic state machine	14	20	13	7	3	56	28
(b) decoder and arithmetic unit	23	20	41	23	6	89	30
(c) input/output unit	27	45	54	25	7	102	30

(a)

```
if (c<6) c<=c+1
else c<=0
if (c<w) pwm<=1
else pwm<=0
```

(b)

```
architecture RTL of pwm is
 signal c : unsigned(2 downto 0) := "000";
begin
process(clk)
begin
 if rising_edge(clk) then
  if c < 6 then
     c <= c + 1;
  else
     c <= to_unsigned(0,3);
  end if;
  if c < w then
     pwm <= '1';
  else
     pwm <= '0';
  end if;
 end if;
end process;
end RTL;
```

(c)

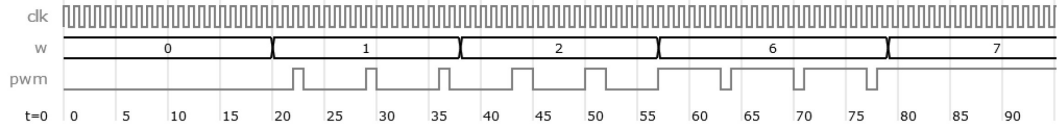

Fig. 13. Pulse width modulator: (a) SHDL model, (b) generated VHDL and (c) simulation.

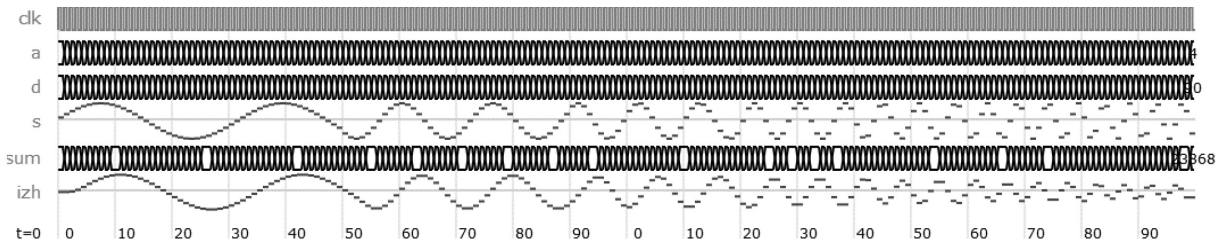

Fig. 14. Simulation waveform of a digital sine sweep generator (s) and FIR filter output (izh).

dents designing components with SHDL can reuse the component instantiation code from the generated test bench to quickly assemble the VHDL structural model.

Design of a small educational central processing unit (CPU) is divided into several steps. The students develop an accumulator-based CPU with 16 machine instructions [24]. We developed an on-line assembler, simulator and a small C compiler to help the students developing CPU hardware and software [25].

The CPU first exercise is a finite state machine control unit used to define the instruction execution stage. The next step is instruction decoding and arithmetic operations logic. The CPU model is tested on the simulator using a read-only memory (ROM) model with small test programs for verification of the implemented instructions. The final exercise is the design of an input/output unit and logic for the corresponding instructions.

3.3 Experience with SHDL and web-tool

3.3.1 Teaching experience

We are teaching the HDL-based digital systems laboratory for 8 years. From the beginning, we used only the FPGA development tools and the VHDL language for the laboratory experiments. The students who already learned VHDL modeling of the basic logic structures requested help regarding language syntax to complete their circuit models. A lot of time was spent on the VHDL syntax issues and the complex tools discouraged the students to install the software at home for self-learning. We considered providing web-tools and simplified hardware description language to improve teaching and learning.

The proposed tool was introduced to laboratory practice for two elective module courses: Integrated circuits and Design of digital electronic systems. The module is taught one semester in the 3rd year of the 1st cycle professional study program. The number of enrolled students is 20–25 each year and they have different pre-knowledge. The students of electronics already learned the VHDL modeling basics, but several students of other programs know only the fundamentals of the digital structures. The students had difficulties to catch up on basic HDL modeling skills due to verbose and complex VHDL syntax rules and unfamiliar design environment. This was our initial motivation to search for educational design entry tools supporting VHDL and finally developing our own SHDL modeling methodology.

Table 3 presents a schedule of the laboratory lessons and a comparison of the exercises in the old and new laboratory program. The students have two to four hours each week to complete the lesson and present designed circuit operation on the simulator or on the programmable prototyping board. At the end, they have three weeks for individual project work.

The lessons begin with an introduction to HDL modeling and simulation tool and combinational logic exercises. The SHDL introduction tutorial described in Chapter 3.1 is used to explain the SHDL modeling basis and web-tool operation. The VHDL syntax and good coding practice are gradually explained on the automatically translated code. Due to the efficiency of the SHDL modeling, students were able to complete more exercises in the same time frame and finish the lessons a week earlier to have more time for the final project. The additional exercises and parts of the lessons completed earlier are in Table 3 denoted with a bold text.

The final project is a VGA game or animation designed by each student on the FPGA development board. The CPU is used in the digital system to control sprites displayed on the VGA monitor. The students upgrade the VGA controller with various sprites and line graphics, connect the controller with the upgraded input/output unit of the CPU and develop game control hardware and software components. Finally, they deliver a short project report and present the designed digital system.

An assessment of the laboratory practice is done by oral examination where the students explain details of the designed system structure and operation, design decisions and possible upgrades. The laboratory project is only a part of the course requirements. When the students complete the project, they receive a grade from the scale: satisfactory, good and excellent. The laboratory project

Table 3. Weekly schedule of elective module laboratory lessons in an old program using VHDL and new program introducing SHDL and VHDL.

Week	Old lessons (VHDL only)	New lessons (SHDL and VHDL)
1	Combinational VHDL introduction: • adder, comparator • decoder, ALE with flags	Combinational HDL introduction: • adder, comparator, **maximum, data sorting** • ALE with flags
2	Sequential basics: • flip-flop, SISO register • clock divider, LED counter	Sequential basics: • SISO register, **LFSR random generator** • clock divider, LED counter
3	Sequential circuits: • clock divider, decoder • microsequencer (semaphore)	Sequential circuits: • divider, **PWM** • microsequencer (**Morse code**)
4	CPU I: • control logic • datapath load, add	CPU I: • control logic, **ROM** • datapath load, add, **conditional jump**
5	CPU II: • program memory: ROM and RAM • data store instruction	CPU II: • RAM, data store, **logic instructions** • **I/O unit**
6	CPU III: • I/O unit • logic or shift or conditional jump	CPU III: • **RAM IP** • **shift** and **conditional carry jump**
7	VGA: • timing generator, test image	VGA: • timing generator, **ROM image**
8	VGA: • ROM image	System: • **CPU, VGA and IO unit integration**
9	System: • CPU and VGA integration	System: • **advanced IO, control software**
10	System: • advanced IO, control software	Individual project work

grades compare the students within the module and do not significantly differ from year to year, but the presented projects in the new program have more digital components, better structure, and less unfinished work.

Recently, we introduced the proposed design tool to a Digital electronics systems course in the 2nd year of the 1st cycle university study program attended by 40 students. The course topics are modeling of digital circuits, introduction to HDL, circuit design and synthesis methodology and digital system design. The laboratory consists of ten two-hour lessons divided into basic HDL exercises and a digital system project. The project goal is to explain the partitioning of a digital system into interface, control, and signal processing components and design the required components. Each year we select a project topic, for example small digital storage oscilloscope, logic analyzer, graphics generator, signal generator or electronic piano. Digital component development according to specifications is a challenge to the students with a little experience in the digital design. Solving the exercise required not only understanding the specification and proposed circuit model, but also developing code in VHDL and use the complex design tools. The students needed a lot of guidance and help in the laboratory. When using our web-tool and SHDL they can perform more modeling experiments on their own. For example, a numerically controlled oscillator used as a piano tone generator is described in SHDL with considerably fewer lines of code compared to VHDL (Table 2). In the same laboratory exercise time-frame, the students can explore different circuit models and better understand the proposed model structure.

We are now able to assign small practical homework exercises which can be solved even on a mobile device. The exercises are focused to teach digital modeling semantics. The exercise questions encourage students to experiment with different solutions, for example:

- How to describe a combinational adder with saturation output and avoid a combinational loop?
- What is the correct way to introduce reset to a sequential modulo counter?

By answering the questions, the students better understand how to design good digital circuit models.

3.3.2 Learning experience

Digital system projects implemented on programmable development boards are motivating the students to experiment and learn logic design using hardware description languages, but the standard languages like VHDL are difficult to learn. We identified the problem of learning efficiency and asked the students about high-level modeling in the VHDL. The students respond that the VHDL language is difficult due to:

- complex and verbose syntax,
- parallel modeling paradigm,
- variety of data type and type conversion rules and
- the lack of simple tools for self-learning.

We received promising feedback when introducing the new tool to the laboratory practice on the digital systems design course. The design exercises specify circuit interface and behavior and not modeling language. The students were able to use both SHDL and VHDL to complete their assignments. Some of the students decided to use only VHDL modeling language, but the majority (85%) used SHDL for the initial experiments and finalize the assignment in the VHDL. We summarized our experience with the new design methodology:

- Most of the students benefit from the SHDL and were able to finish their assignments faster and with less help of the tutor.
- The SHDL can be introduced to the laboratory with minimal overhead due to simple syntax and VHDL-like operators.
- User-friendly web tool allowed more modeling and verification experiments leading to better final designs.
- The students used automatic SHDL to VHDL translation even for upgrading the existing VHDL models where they must merge produced code with the existing one.

4. Conclusions

Hardware description language based digital circuits design is a skill required in the digital electronics industry and consequently an important outcome of the digital design education. We use VHDL in several electrical engineering digital design courses. Digital circuit teaching follows the bottom-up approach, where the students simultaneously learn basic digital structures and their VHDL models. In the digital systems design laboratory, we experienced problems with the complexity of the standard language VHDL and related development tools. A considerable amount of time was spent repeating the language syntax rules and helping the students to produce correct models.

To improve the teaching experience, we developed a digital modeling methodology based on the small hardware description language and the corresponding web tool. The new tool is used for high-level circuit design, simulation and efficient development of digital system components. The substantially simplified modeling language syntax enables quick introduction to laboratory practice. The typical laboratory exercise models have in SHDL 4-times fewer lines of code which enables the students to develop more circuits. The web tool producing properly formatted VHDL code can be seamlessly integrated into the VHDL modeling and prototyping design flow.

The students learned the HDL circuit design with the new methodology faster and finished their assignments on their own. Their final projects have more components and better structure. The freely available user-friendly web tool can be used in self-learning experiments. In the future, we plan to connect the tool with assignments database and automatic simulation response to further encourage online learning.

References

1. W. J. Dally, R. C. Harting and T. M. Aamodt, *Digital Design Using VHDL*, Cambridge University Press, Cambridge, 2016.
2. J. F. Wakerly, *Digital Design: Principles and Practices*, 5th Edition, Pearson, New York, 2018.
3. H. A. Ochoa and M. V. Shirvaikar, A Survey of Digital Systems Curriculum and Pedagogy in Electrical and Computer Engineering Programs, *ASEE Gulf-Southwest Section Annual Conference*, Austin, 2018.
4. B. Nikolic, Z. Radivojevic, J. Djordjevic and Veljko Milutinovic: A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization, *IEEE Transactions on Education*, **52**(4), pp. 449–458, 2009.
5. D. Hanna and R. E. Haskell, Learning Digital Systems Design in VHDL by Example in a Junior Course, *Proceedings of the ASEE North Central Section Conference*, Charleston, March 30–31, 2007.
6. D. Zhang, P. Qian, L. Wang, Y. Guo and C. Sun, Experimental Case Design of Digital Logic Based on Through-Type Teaching, *13th International Conference on Computer Science & Education (ICCSE)*, Colombo, pp. 552–555, 2018.
7. F. Machado, S. Borromeo and N. Malpica (Madrid), Project Based Learning Experience in VHDL Digital Electronic Circuit Design, *IEEE International Conference on Microelectronic Systems Education*, San Francisco, pp. 49–52, 2009.
8. J. M. P. Cardoso, A teaching strategy for developing application specific architectures for FPGAs, *International Journal of Engineering Education*, **24**(4), pp. 833–842, 2008.
9. A. Trost and A. Žemva, Teaching design of video processing circuits, *International Journal of Electrical Engineering Education*, **49**(2), pp. 170–178, 2012.
10. O. B. Adamo, P. Guturu and M. R. Varanasi, An innovative method of teaching digital system design in an undergraduate electrical and computer engineering curriculum, *IEEE International Conference on Microelectronic Systems Education*, San Francisco, 2009.
11. E. Magdaleno, M. Rodríguez, D. Hernandez, E Rodrigues and F. Perez, Teaching methodology of the subject design of electronic systems using FPGA in the new master of indus-

trial engineering, *Technologies Applied to Electronics Teaching (TAEE)*, Seville, pp. 1–6, 2016.
12. Xilinx Inc., Vivado Design Suite—HLx Editions, Available from: https://www.xilinx.com/products/design-tools/vivado.html, Accessed 14 January 2019.
13. Z. Stanisavljevic, V. Pavlovic, B. Nikolic and J. Djordjevic, SDLDS—System for Digital Logic Design and Simulation, *IEEE Transactions on Education*, **56**(2), pp. 235–245, 2013.
14. Zubia, WebLab-Deusto-CPLD: A Practical Experience, International Journal of Online Engineering, Wien, pp. 17–18, 2012.
15. Y. Ding and S. Li, A Web-Based System for Digital Logic Experiments, *International Conference on Computer Science & Education (ICCSE)*, Colombo, pp. 800–803, 2018.
16. A. Kumar, R. C. Panicker and A. Kassim, Enhancing VHDL Learning through a Light-weight Integrated Environment for Development and Automated Checking, *IEEE International Conference on Teaching, Assessment and Learning for Engineering*, Bali, pp. 570–575, 2013.
17. G. R. Garay, A. Tchernykh, A. Yu. Drozdov, S. N. Garichev, S. Nesmachnow and M. Torres-Martinez, Visualization of VHDL-based simulations as a pedagogical tool for supporting computer science education, *Journal of Computational Science*, in press, Available online 2017.
18. G. Wang, Lessons and Experiences of Teaching VHDL, *American Society for Engineering Education Annual Conference*, Honolulu, 2007.
19. W. Meeus, K. Van Beeck, T. Goedeme, J. Meel and D. Stroobandt, An overview of today's high-level synthesis tools, *Design Automation for Embedded Systems*, **16**(3), 2012, pp. 31–51.
20. S. Schocken, N. Nisan and M Armoni, A synthesis course in hardware architecture, compilers, and software engineering, *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, New York, pp. 443–447, 2009.
21. K. Becker, A web based tool for teaching hardware design based on the plain simple hardware description language, EDUCON, Istanbul, pp. 88–93, 2014.
22. SHDL online, http://lniv.fe.uni-lj.si/shdl/, Accessed 14 January 2019.
23. High-level logic modeling JavaScript sources, https://github.com/andrejtrost/dig-model-sim, Accessed 14 January 2019.
24. A. Trost and A. Žemva, Design of custom processors for the FPGA devices, *Elektrotehniški vestnik*, **79**(1–2), pp. 55–60, 2012.
25. LNIV Virtual LAB, http://lniv.fe.uni-lj.si/cpu.html, Accessed 14 January 2019.

Andrej Trost received his PhD degree in 2000 from the Faculty of Electrical Engineering, University of Ljubljana. Currently he works at the same faculty as an associate professor teaching high-level design techniques on several graduate and post-graduate study levels. His research interests include the FPGA technology and digital systems design for academic and industrial applications.

Andrej Žemva received his BSc, MSc and PhD degrees in electrical engineering from the University of Ljubljana in 1989, 1993 and 1996, respectively. He is Professor at the Faculty of Electrical Engineering. His current research interests include digital signal processing, HW/SW co-design, ECG signal analysis, logic synthesis and optimization, test pattern generation and fault modeling.