ELEKTROTEHNIŠKI VESTNIK 89(3): 124–132, 2022
ORIGINAL SCIENTIFIC PAPER

Online Digital-Circuit Modeling with Data-Flow Visualisation and Area Estimation

Andrej Trost[1], **Manfred Ley**[2], **Andrej Žemva**[1]

[1]*University of Ljubljana, Faculty of Electrical Engineering, Tržaška 25, 1000 Ljubljana, Slovenia*
[2]*Carinthia University of Applied Sciences - Engineering IT, Villach, Austria*

† *E-mail: andrej.trost@fe.uni-lj.si*

Abstract. Digital circuits are efficiently designed with abstract models in hardware description languages. Digital design which requires understanding the modeling language, design-flow and tools is considered difficult to the entry-level students. To boost learning, we propose a small hardware description language (SHDL) and an online tool for modeling, simulation and transformation to a standard language. The paper presents the SHDL structure and a novel tool for data-flow visualisation and circut area estimation. Early estimation of the synthesized circuit structure helps students at their taking circuit modeling design decisions.

Keywords: digital circuit model, high-level language, data flow graph, circuit area estimation, online tool

Spletno modeliranje digitalnih vezij s prikazom podatkovnega toka in oceno površine

Digitalna vezja učinkovito načrtujemo z abstraktnimi modeli v strojno-opisnem jeziku. Proces zasnove vezja, ki zahteva poznavanje novega jezika, postopkov in orodij, je za študente nižjih letnikov zelo zahteven. Poučevanje lahko izboljšamo s poenostavljenim modelirnim jezikom SHDL in spletnim orodjem za opis, simulacijo ter pretvorbo modela v standardni jezik. V članku predstavljamo zgradbo jezika SHDL in novo orodje za prikaz gradnikov podatkovnega toka in oceno površine vezja. Ocena zgradbe sinteziranega vezja pomaga študentom pri načrtovalskih odločitvah v modelu vezja.

1 INTRODUCTION

Digital circuit modeling in a hardware description language (HDL) is one of the main tasks in the digital design process. Standard languages VHDL and Verilog are used in the circuit modeling of the front-end as well the as back-end logic synthesis tools. A VHDL model describing circuit components operating in parallel follows different semantic rules compared to a typical computer language [1], [2].

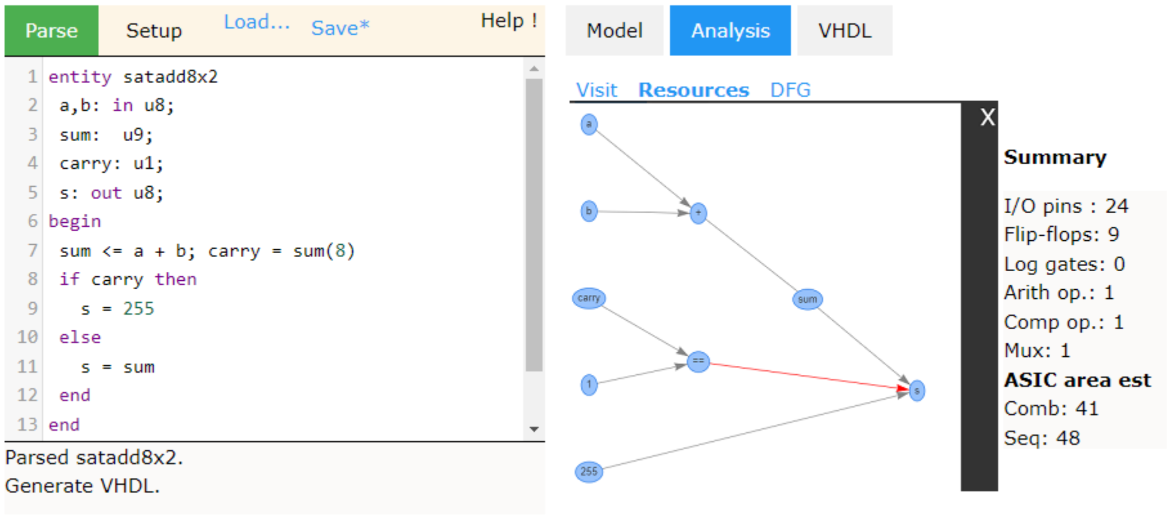
Designing digital circuits requires knowledge of logic and registers operation, HDL syntax and design tools. The languages were initially developed for simulation and only a language subset is used when describing synthesizable models. The standard language modeling methodology is considered difficult to unexperienced designers.

Received 24 May 2022
Accepted 23 June 2022

The availability of computer aided tools is crucial for a widespread adoption of the HDL design methodology. The back-end synthesis tools are provided by programmable device vendors, since they are tightly coupled with the technology implementation process. These tools provide also front-end modeling and simulation environment. For example, the Xilinx Vivado Design Suite [4] accepts a register-transfer level (RTL) HDL and higher-level circuit models, but their usage requires a substantial training. The designer learning VHDL first needs a free and simple-to-use circuit simulator. GHDL [4] is a command-line simulator which needs an additional software for viewing simulation waveforms. Researchers develop a lightweight design environment [5] and distributed online VHDL compiler and simulator [6]. The EDA Playground [7] provides an online tool for testing various digital development tools and languages. The tools require a VHDL design as well as a test-bench file for simulation.

Digital circuit design on a higher-level of abstraction is aimed to boost the design efficiency, specifically for the development of hardware algorithm accelerators. High-level languages, such as Bluespec [8], Chisel [9], OpenCL [10] and synthesis tools [11], address these needs. A good understanding of the RTL circuit models is still required to effectively use high-level tools. The RTL languages and models provide the designer a full control of the hardware structure and enable an efficient gate-level technology optimization.

Simplified hardware description languages and associated tools have been proposed to help learning the HDL design methodology. A plain simple HDL with a web



```
entity satadd8x2
 a,b: in u8;
 sum:  u9;
 carry: u1;
 s: out u8;
begin
 sum <= a + b; carry = sum(8)
 if carry then
   s = 255
 else
   s = sum
 end
end
```

```
Parsed satadd8x2.
Generate VHDL.
```

```
Summary
I/O pins : 24
Flip-flops: 9
Log gates: 0
Arith op.: 1
Comp op.: 1
Mux: 1
ASIC area est
Comb: 41
Seq: 48
```

Figure 1. Online SHDL tool with an example SHDL model, dataflow graph and resource estimation summary.

tool [12] introduces hardware modeling in the C-like syntax to help students with an unfamiliar HDL syntax and programming paradigm. A Finite State Machine modeling language and tools [13] enable fast prototyping for a specific type of digital circuits. CompactHDL [14] introduces a simplified version of VHDL and Java tools for an automatic translation to VHDL.

We present a Small Hardware Description Language (SHDL) and an associated online tool to be used in digital design education [15]. The tool includes a circuit simulator and outputs a nicely formatted VHDL code which enables the designer to learn and adopt a proper VHDL modeling practice. Recently, the tool was upgraded with a circuit area estimation from the high-level SHDL model [18]. The estimated circuit area calculation provides a quick feedback to the designer without using an external synthesis tool. Figure 1 depicts our online SHDL tool with a code editor, circuit graph and resource estimator.

Chapter 2 describes the used high-level circuit description language and its connection to the standard VHDL. Chapter 3 describes our SHDL model parsing and resource estimation methodology. Chapter 4 discusses the use of SHDL in education and plans for our future work.

2 High-Level Circuit Description

Hardware description languages model digital circuits on structural, data-flow and behavioral abstraction levels. Structural models describe circuit schematics with signal declarations and component connections (instantiation). High-level HDL models define combinational logic with operators in concurrent assignment statements instead of logic components and gates. The statements order is not important, because the circuit structure is derived from the flow of the data signals between expression operators and assignments. Synthesis constraints are applied to the signal data type and operator support. Binary vectors representing signed or unsigned integers and basic operators are extensively supported, but real numbers should be avoided in a synthesizable model.

Behavioral modeling in HDL is used to describe the circuit operation in terms of an algorithm. The basic language constructs for an algorithm specification are: assignments, conditional statements and loops. A process block with a sequence of statements is used in VHDL for a behavioral model. The process is executed in an infinite loop presenting an iterative circuit behavior, e.g. counters or finite-state machines. The models are synthesizable considering several restrictions: specific usage of loops and describing the synchronous logic.

2.1 Small Hardware Description Language

Hardware description languages contain data types and structures to describe circuit models. Synthesizable models use only a subset of a standard hardware description language. We propose to further simplify the modeling language to boost learning the HDL design [16]. Figure 2 presents the proposed SHDL structures and composition of a circuit model.

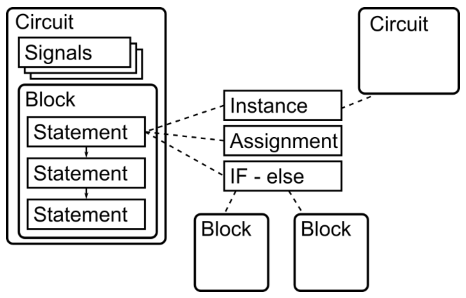


Figure 2. SHDL constructs for building a digital circuit model.

A digital circuit model contains a set of signal declarations and a function specification block with a sequence of statements. The basic concepts of the structural, data-flow and behavioral models are described with the following three statement types:

- instance – a component instantiation,
- assignment – describes the combinational data-flow as well as synchronous data storage components,
- conditional (if-else) statement for the algorithmic behavioral specification.

The SHDL syntax is similar to the VHDL. The proposed language syntax rules can be expressed in the Backus-Naur Form (BNF):

```
circuit :== ['entity' name] {declaration}
            ['begin'] block ['end']
declaration :== name_list ':' ['in' | 'out'] type
name_list   :== identifier {, identifier}
block       :== statement {[;] statement}
statement   :== assign | instance | if_statement
assign      :== identifier assign_op expression
assign_op   :== '=' | '<='
instance    :== identifier '(' name_list ')'
if_statement:== 'if' (condition) 'then' block
            {'elsif' block} ['else' block] 'end'
```

A circuit model contains an entity name, declaration of ports and internal signals, and statement blocks. The basic data types are an one-bit signal and multi-bit vector presenting a bus. The data values on the bus can be treated as signed or unsigned integers.

Signals and constant literals are combined with the Boolean logic, vector shift and basic arithmetic operators in the SHDL assignment expressions. The expressions of the BNF syntax rules are recursively defined as:

```
expression :== bool { or | xor | xnor bool}
bool       :== relation { and relation }
relation   :== shift {relation_operator} shift
shift      :== simpleExp {sll, srl literal}
simpleExp  :== term {+ |- | & term}
term       :== factor {* factor}
factor     :== primary |- primary |not primary
primary    :== name | literal | (expression)
```

2.2 *SHDL circuit examples*

An example of an 8-bit adder model with a carry input in the proposed SHDL:

```
entity add8
 a,b: in u8
 ci: in u1
 s: out u9
begin
 s = a + b + ci
end
```

The SHDL basic data types are: u1 for one-bit signals, uN for N-bit unsigned and sN for N-bit signed vectors. The language keywords are similar to those of a standard VHDL, but the description is less verbose without libraries and architecture section. VHDL requires a perfect data-type match in assignments obtained by resizing and type conversion functions. A model of the same adder in VHDL:

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

entity add8 is
 port (
   a,b: in unsigned(7 downto 0);
   ci : in std_logic;
   s  : out unsigned(8 downto 0) );
end add8;

architecture logic of add8 is
begin
s <= (resize(a,9) + resize(b,9)) +
      unsigned'("" & ci);
end logic;
```

Our functional circuit model is composed of concurrent assignment statements between keywords **begin** and **end**. SHDL supports similar basic arithmetic and logic operations on signals as VHDL: vector addition, subtraction, multiplication and Boolean operations. The hardware specific vector operations are concatenation (&) and slicing.

Constant vector slicing describes a combinational truth table or decoder in case of a vector array. A 7-segment decoder example declares the rom array with ten 7-bit unsigned binary values (data type 10u7):

```
entity decod
 bcd: in u4
 led: out u7
 rom: 10u7= "0111111","0000110","1011011",
  "1001111","1100110","1101101","1111101",
  "0000111","1111111","1101111";
begin
 led = rom(bcd)
end
```

If the vector is not constant, indexing with another vector describes the multiplexer. Example of a 16-to-1 multiplexer:

```
entity mux
 d: in u16;
 sel: in u4;
 y: out u1;
begin
 y = d(sel)
end
```

A two-input multiplexer is described with a conditional assignment (when-else) statement. The conditionally selected expressions describe the data-flow logic with various combinational components. For example:

```
y = a+b when b>0 else a-b
```

describes the circuit with an adder, subtractor, comparator and two-input multiplexer (see Figure 3).

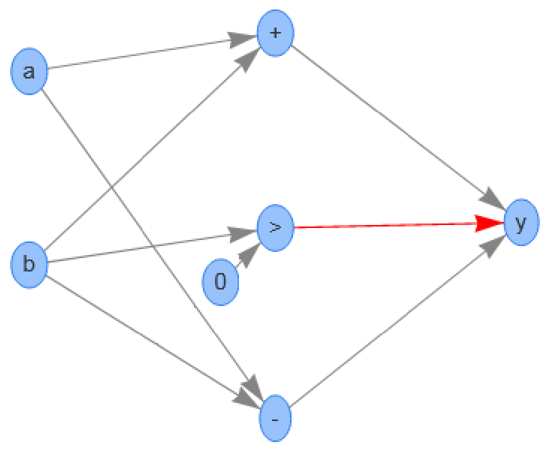


Figure 3. Combinational data flow graph.

The assignment operator <= describes in the SHDL sequential logic, where the assignments are executed at the rising edge of the system clock. The sequential circuit models are constrained to a synchronous logic with a single clock, which is sufficient for small educational components. An accumulator with reset and clock enable signals in the SHDL:

```
 if reset then
  a <= 0
 elsif en=1 then
  a <= a + d
 end
```

The same circuit in VHDL requires a process with a rising clock condition. In SHDL, a signal condition is either a signal value (0 is false) or expressed with a logic relation (e.g. en=1). The constant values are specified as integer numbers as opposed to VHDL where a binary notation or integer conversion functions should be applied:

```
process(clk)
begin
 if rising_edge(clk) then
  if reset = '1' then
     a <= to_unsigned(0, 8);
  elsif en = '1' then
     a <= a + d;
  end if;
 end if;
end process;
```

3 Parsing and Resource Estimation

Our SHDL parser is implemented as an open-source tool for a web browser. The SHDL web page is designed in HTML5 for a responsive, user-friendly utilization and it uses a set of the JavaScript library files for the model parsing, simulation, conversion to VHDL and resource estimation. The web page is divided into three sections: SHDL editor with a parser log on the left side, model settings and outputs on the right side (see Figure 1) and interactive simulator at the bottom.

3.1 Editor and parser

The editor is based on an open source project CodeMirror [19] which offers VHDL syntax coloring. The SHDL modeling structures are implemented in a library *model.js* with the JavaScript function closure objects:

- NumConst: numeric constants,
- Var and Slice: signal variables and bit slices,
- Op: recursive binary expressions,
- Statement: assignment statements,
- IfStatemet: conditional statement blocks,
- Instance: model instances and
- Block: SHDL statements block.

The objects define a set of methods used for accessing the internal data, visiting and analyzing the model, evaluating the circuit model for the simulation purpose and producing a standard HDL output. A library named *vector.js* is used for numeric calculations with up to 64-bit signed or unsigned values.

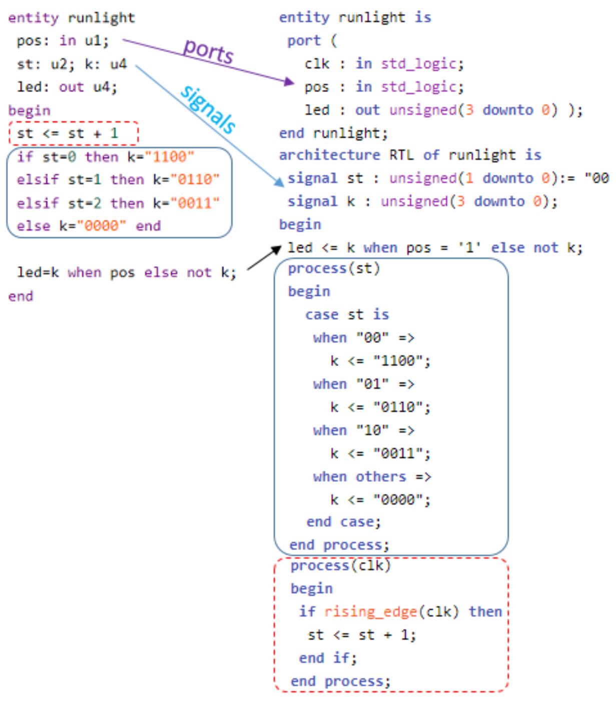
A *parsesim.js* library is used to translate the SHDL code to the modeling structures. The input code is first processed by a lexical analyzer (*lexer.js*) producing basic language tokens. The parser reads the tokens and builds the circuit model according to the syntax rules. The parsing process stops in case of a rule violation and outputs an error log. During expression parsing for the assignment and conditional statements, a data type of every operation object is recursively calculated by the operator and operands data type.

For every assignment statement in a code block, the assignment target variable name is stored in a list of targets to alert the user in case of multiple assignments to the same variable in a code block.

3.2 VHDL output and interactive simulation

Figure 4 presents our SHDL model of a sequential circuit converted into the VHDL model. Entity section of the VHDL model contains a port signal declaration and inferred clock. The internal signals are declared in the architecture section. The signals which are an output of the sequential logic are initialized to zero unless there is a specific value assigned in SHDL.

Sequential assigments in SHDL are transformed to a VHDL clocked process. Conditional statements with combinational assignments are converted into a combinational process. A sequence of conditional statements where a signal is compared to a set of constants can be transformed to a case statement in VHDL.



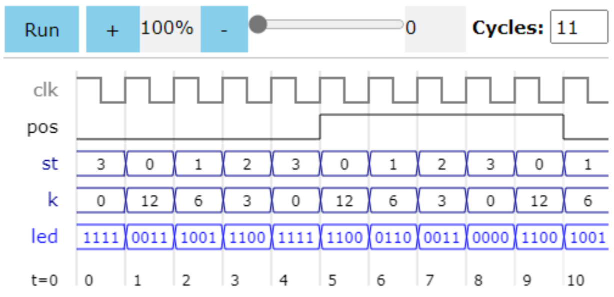
```
entity runlight
 pos: in u1;
 st: u2; k: u4
 led: out u4;
begin
st <= st + 1
 if st=0 then k="1100"
 elsif st=1 then k="0110"
 elsif st=2 then k="0011"
 else k="0000" end

 led=k when pos else not k;
end
```

```
entity runlight is
 port (
   clk : in std_logic;
   pos : in std_logic;
   led : out unsigned(3 downto 0) );
end runlight;
architecture RTL of runlight is
 signal st : unsigned(1 downto 0):= "00
 signal k : unsigned(3 downto 0);
begin
 led <= k when pos = '1' else not k;
 process(st)
 begin
   case st is
    when "00" =>
      k <= "1100";
    when "01" =>
      k <= "0110";
    when "10" =>
      k <= "0011";
    when others =>
      k <= "0000";
   end case;
 end process;
 process(clk)
 begin
  if rising_edge(clk) then
   st <= st + 1;
  end if;
 end process;
```

Figure 4. Conversion from SHDL to VHDL.

The online tool includes an interactive model simulator utilizing computation library *vector.js* and waveform display library *wave.js*. The simulator reads the input signal values set by the user and preforms a repeated model evaluation and waveform updating. The simulation setup is used to generate a VHDL test-bench for connection with external tools.

A discrete event simulation is executed by visiting and evaluating the model in a sequence of delta simulation cycles. At each cycle, the simulator computes events for the assignment target signals and updates the signal values at the end of the cycle. The simulation result is displayed on the waveform presented in Figure 5.



```
st   3    0    1    2    3    0    1    2    3    0    1
k    0    12   6    3    0    12   6    3    0    12   6
led  1111 0011 1001 1100 1111 1100 0110 0011 0000 1100 1001
t=0  0    1    2    3    4    5    6    7    8    9    10
```

Figure 5. Interactive SHDL simulator.

3.3 Circuit area estimation

A high-level SHDL model is decomposed into basic combinational and sequential digital circuit building blocks to evaluate the circuit in terms of the occupied silicon area. The actual circuit area is obtained after synthesis of the circuit model targeting the selected technology. The CMOS synthesis tools are expensive and not easily accessible to students learning the digital design, so an estimation of the circuit area is included in our online SHDL tool.

The area estimation is based on characterisation of the sample circuit models in a selected CMOS technology. Digital building blocks described with the SHDL operators and modeling constructs are identified from the high-level model. The circuit area estimation process is described on an example of a pulse width modulator (PWM) circuit. The circuit has an 8-bit signed input **data** and one bit output **pwm**. PWM is composed of a counter and comparator. The 8-bit counter is described by signal **c** counting from 0 to 254. When the counter is reset, the input **data** is transformed to the unsigned by adding offset 128 and then saved to internal register **d**. The register is finally compared to internal counter **c** to obtain a one-bit pulse output:

```
entity pwm8
 data: in s8
pwm: out u1
 c, d: u8
begin
 if c=254 then
  c <= 0; d <= data+128
 else
  c <= c+1
 end
pwm = 1 when c<d else 0
end
```

The SHDL model parser transforms the behavioral model of the sequential circuit into the RTL model, where all flip-flops and registers are separated from expressions and conditional statements. The transformation requires creating new internal signals (c_next, d_next) for the detected sequential signals. The new signals are part of the combinational logic description which begins by assigning the default signal values:

```
d_next = d; c_next = c
 if c = 254 then
  c_next = 0; d_next = (data + 128)
else
  c_next = (c + 1)
end

 if c < d then
 pwm = 1
else
 pwm = 0
end
-- separated registers
c <= c_next; d <= d_next
```

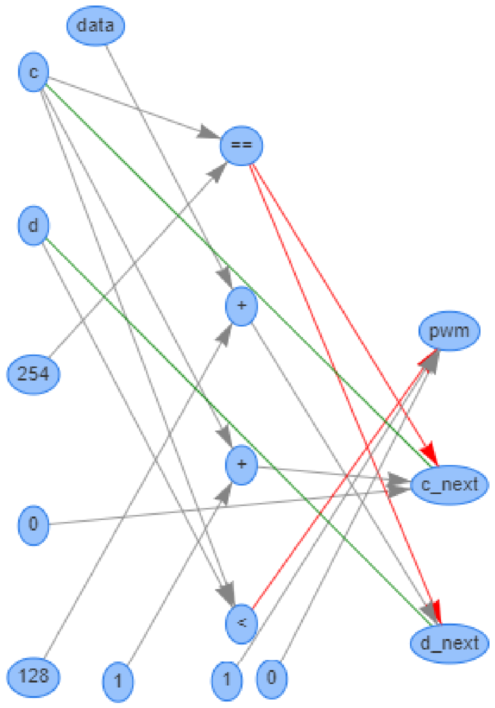


Figure 6. PWM data-flow graph.

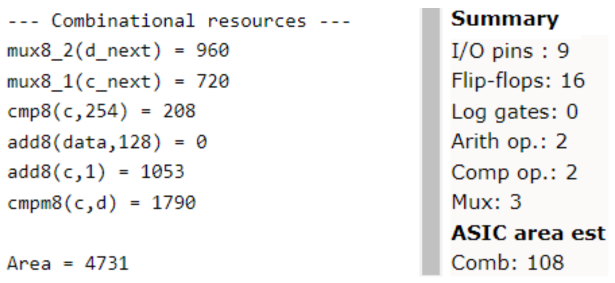
A high-level description of the combinational logic is used to construct a data-flow graph (DFG) visiting the parsed SHDL model. The data-flow expressions are recursively visited to produce DFG nodes and edges. The DFG nodes represent expression operators, their inputs and assigned signals.

Figure 6 shows DFG for the combinational logic of the PWM circuit example. A behavioral code with conditional statements produces multiplexer nodes with more than two inputs. The control multiplexer input from the conditional expression is denoted in DFG by a red line. The constructed DFG visualization based on the vis-network [20] is added to the SHDL web tool for the debugging purpose.

The DFG nodes are tagged with the information required by the area estimation, for example the size and type of the input and output edges. The estimation algorithm finally traverses the DFG calling area estimation functions based on node tags. These functions calculate estimated circuit area for a specified building block.

Combinational building blocks for the PWM example are listed in Figure 7: 8-bit multiplexers mux8 for c_next and d_next, 8-bit comparator cmp8, two 8-bit adders add8 and one 8-bit magnitude comparator cmpm8. The blok names include tags to give an additional information for the area estimation: e.g. the $mux8\_2(d\_next)$ is an 8-bit 2-input multipexer and $mux8\_1(c\_next)$ is an 8-bit multiplexer with one vector data input and one constant input (c_next = 0).

The estimated combinational logic area is a sum of



```
--- Combinational resources ---
mux8_2(d_next) = 960
mux8_1(c_next) = 720
cmp8(c,254) = 208
add8(data,128) = 0
add8(c,1) = 1053
cmpm8(c,d) = 1790

Area = 4731
```

Summary
I/O pins : 9
Flip-flops: 16
Log gates: 0
Arith op.: 2
Comp op.: 2
Mux: 3
ASIC area est
Comb: 108

Figure 7. PWM resource usage estimation.

the estimated block areas. The non-combinational circuit area is calculated by counting the number of flip-flops multiplied by the sythesized flip-flop size.

3.4 *Area estimation functions*

The circuit area for combinational blocks is estimated as a function of the input size and tags. The circuit area analysis is performed on several sets of combinational circuits generated by the SHDL models. The circuits are synthesized by Synopsys ASIC DesignCompiler using an adapted 0.35μm CMOS technology library. The adapted library contains basic logic gates with a maximum of four inputs. The only complex structures are full (FA) and half adders (HA). By disabling complex gates and using default area optimization, the synthesized circuit structure is assumed to be similar to lecture book circuits, where the building blocks size is estimated with linear functions of the inputs size.

Figure 8 presents an example of the iterative circuit structure for ripple-carry adders (RCA) with the same sized inputs, different sized inputs and adders with a constant input. The circuits are composed of standard adders (HA, FA) and some optimized (HA1, HA2, HA3) due to constant inputs and no carry output (FA1).

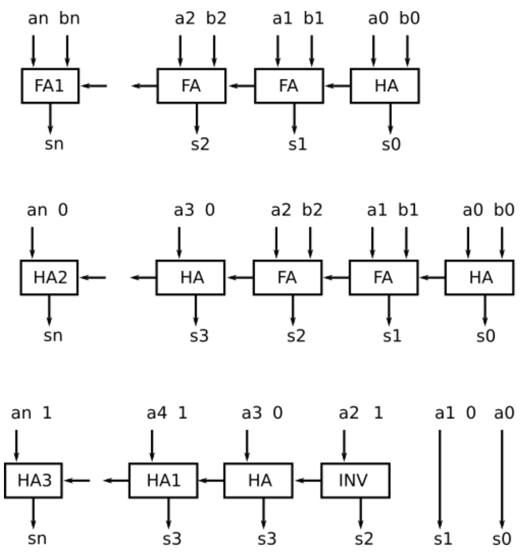


Figure 8. Ripple-carry adders: an + bn, am + bn, an + const

The n-bit RCA circuit area is a sum of one HA, one FA1 and $n - 2$ FA circuit areas. When one of the operands is smaller vector, the area equation should consider both vector sizes. When one the inputs is constant, the circuit structure depends on a constant value. If the least significant portion of the constant is zero, the optimizer completely removes the logic gates for this part of the adder (see Figure 8).

To verify the assumption of the linear scaling of the arithmetic circuit size, a set of vector adders and subtractors is generated and synthesized. Figure 9 shows la inear dependency of the synthesized adder and subtractor circuit area on the input vector size ranging from 2 to 64 bits.

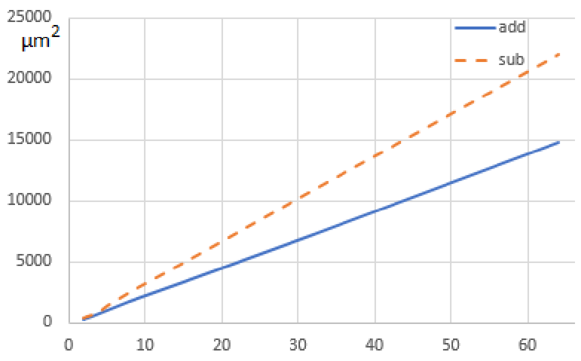


Figure 9. Synthesized adder (solid) and subtractor (dotted) logic area for 2- to 64-bit vectors.

The reported synthesis areas are used to write an overdetermined system of linear equations. The solutions are linear area estimation functions which produces a maximum of a 0.5% error in the approximated size for the adder and 5.3% for the subtractor circuits [18].

The similar methodology is used for multiplication, which scales with a square of the input size and operations on input vectors. Table 1 presents the area estimation functions for the basic logic and arithmetic operators.

Table 1. Area estimation for the arithmetic and logic operators.

Operator	Estimated area [μm^2]
and, or	$A_{ao} \approx 58n$
xor, xnor	$A_{xo} \approx 87n$
add	$A_{add} \approx 233n - 148$
subtract	$A_{sub} \approx 350n - 303$
multiply	$A_{mul} \approx 254n^2 - 691$

Combinational circuits for relational operators can be divided into smaller equality comparators and larger magnitude comparators (see Figure 10). The vector equality or non-equality comparator circuits have an almost identical area which is scaled linearly with the vector size. Similarly, this applies to the magnitude comparators, where the circuit area scales with the input vector size and can be estimated independently of the operator type.

Our analysis of the synthesized area reveals that the scaling is slightly different for smaller magnitude comparators. Better area estimation results are obtained by dividing the linear scaling function into two segments. For example, the segments of comparator $a < b$, are: $250n - 210, n \leq 8$ and $263n - 685$.

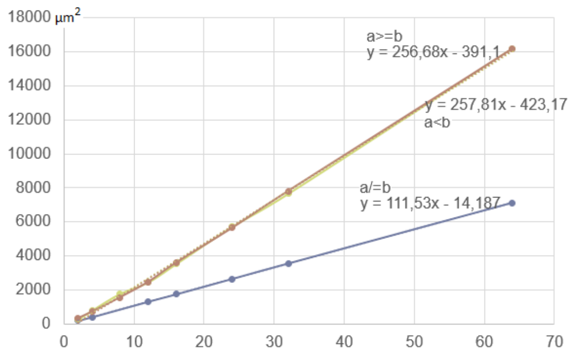


Figure 10. Synthesized magnitude comparators area.

The SHDL expressions can have also constant values and different vector sizes producing smaller circuits due to logic optimizations, as described in the case of the ripple-carry adders. To get better estimation functions for these cases, several sets of combinational circuits are synthesized and multi-parameter area estimation functions are calculated. For example, the area of an n-bit adder with the m-bit input (m<n) is: $A_{addm} \approx 140m + 93n - 142$ and with a constant input: $A_{addc} \approx 161n - 157t - 235$, where t is a number of trailing zero bits in the constant.

Figure 11 presents the circuit area of 12-bit comparators with eight different input constants. The comparator area ranges from 220 μm to 740 μm and can not be predicted with a linear estimation due to optimization of the logic included in the synthesis tool. Our area estimation predicts only the upper bound of the circuit area which is 792 μm for the 12-bit input. The prediction assumes an iterative comparator circuit structure composed of a series of nand/nor logic gates and inverters.

Multiplexers are extracted from multiple assignments to the same signal. The multiplexer area depends on the number (n) and width (d) of the data inputs: $A_{mux} \approx (96n - 26) \cdot d$.

Sequential circuits in the SHDL models are composed of synchronous flip-flops. The non-combinational circuit area is estimated by counting the number of D the flip-flops multiplied by an average flip-flop area.

The total estimated circuit area is a sum of combinational and sequential logic areas. The numbers are normalized to a standard 2-input nand gate size and available in the resource summary (Figure 7).

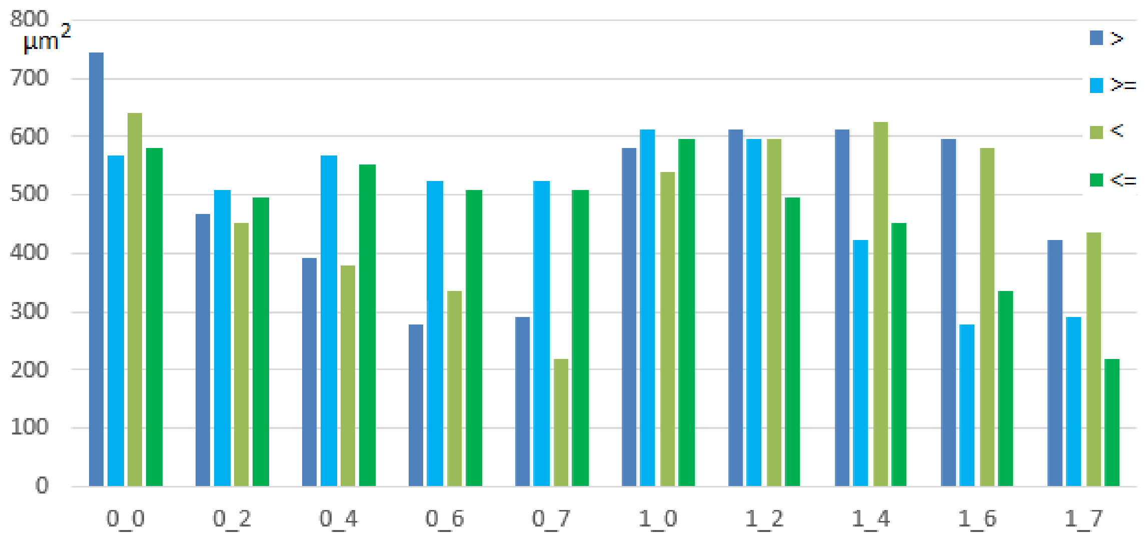


Figure 11. Area of 12-bit constant magnitude comparators.

4 Discussion and Conclusions

We presente an online open source SHDL design tool with model analysis and automatic conversion to standard VHDL. The SHDL modeling language is used to lead unexperienced digital designers in the first steps of the digital circuit data-flow and behavioral modeling. The proposed language syntax is a trade-off between standard HDL, which covers a lot of digital modeling aspects, and modeling complexity. Our goal is to make a common case simple, for example describing a synchronous counter in one line of the code. The SHDL tool outputs a clearly formatted VHDL code suitable for back-end programmable devices implementation tools.

The online tool enables students to practice the digital design in laboratory and at home on any computer platform. It is used in entry-level digital design courses laboratory practice for modeling smaller combinational and sequential circuits: counters, digital modulators, numerical oscillators, interfaces, display controllers and even small CPU models. Most of the example educational circuits previously developed in VHDL are now for the first time designed with SHDL. Learning digital modeling in SHDL uses the generic HDL semantics:

- concurrent description of the combinational circuits data-flow,
- digital operator behaviour (sign, overflow),
- covering all cases in combinational circuits,
- modeling storage elements and
- sequential logic with a feedback loop.

The simplified syntax enables students to carry out more exercises and improve their digital design knowledge. The paper presents our SHDL tool upgraded with data-flow visualisation and circuit area estimation. The circuit area is associated with the cost of the implemented circuit. The exact numbers are obtained after the model synthesizing and optimizing for the target technology, but an early estimation of the area supports designers to make proper model description decisions.

We plan to further improve the circuit model performance estimation and to support other HDL modeling and verification structures. The SHDL tool has a potential of being upgraded to an intellectual property circuit generator tool. We are also considering simplified Verilog as an optional SHDL design and output language.

Acknowledgement

This work was supported by the Slovenian Research Agency (research program grant numbers P2-0197 and P2-0415).

References

[1] M. M. R. Mano, M. D. Ciletti, “Digital Design, Global Edition”, Pearson Education Limited, Harlow, UK, 2018.

[2] W. J. Dally, R. C. Harting, T. M. Aamodt, “Digital Design Using VHDL”, Cambridge University Press, Cambridge, 2016

[3] Xilinx Inc., “Vivado Design Suite – HLx Editions”, Available from: https://www.xilinx.com/ /products/design-tools/vivado.html

[4] T. Gingold, GHDL, 2017, Available from: http://ghdl.free.fr/

[5] A. Kumar, R. C. Panicker and A. Kassim, ”Enhancing VHDL learning through a light-weight integrated environment for development and automated checking,” Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bali, 2013, pp. 570-575.

[6] M. Dasygenis, ”A distributed VHDL compiler and simulator accessible from the web,” 2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Palma de Mallorca, 2014, pp. 1-7

[7] Doulos, EDA Playground Documentation, July 2018, Available from: http://www.edaplayground.com/

[8] M. Arvind, Bluespec: “A Language for hardware design, simulation, synthesis and verification”, Extended Abstract, In Proceedings of MEMOCODE1, ACM, June 2003

[9] J. Bachrach et al., “Chisel: Constructing hardware in a Scala embedded language”, DAC Design Automation Conference 2012, San Francisco, CA, 2012, pp. 1212-1221.

[10] Khronos OpenCL Overview, Available from: https://www.khronos.org /opencl/

[11] W. Meeus, K. Van Beeck, T. Goedeme, J. Meel, D. Stroobandt, “An overview of today’s high-level synthesis tools”, Design Automation for Embedded Systems, 16(3), 2012, pp. 31 – 51

[12] K. Becker, “A web based tool for teaching hardware design based on the plain simple hardware description language”, EDUCON, Istanbul, 2014, pp. 88-93

[13] B. Vandeportaele, ”A Finite State Machine modeling language and the associated tools allowing fast prototyping for FPGA devices,” 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), Donostia-San Sebastian, 2017, pp. 1-6.

[14] F. M. Birleanu, ”CHDL1: Implementing a simplified version of the CompactHDL hardware description language”, Journal of Electrical Engineering, Electronics, Control and Computer Science – JEEECCS, Volume 4, Issue 14, pages 17-22, 2018.

[15] A. Trost and A. Žemva, ”Online VHDL Generator and Analysis Tool,” 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2019, pp. 1-4.

[16] High-level logic modeling JavaScript sources, 2019, Available from: https://github.com/andrejtrost/dig-model-sim

[17] A. Trost and A. Žemva, ”A web-based tool for learning digital circuit high-level modeling”, International journal of engineering education, 2019, vol. 35, no. 4, pp. 1224-1237

[18] A. Trost, M. Ley, ”High-level circuit model area estimation”, 56th International Conference on Microelectronics, Devices and Materials the Workshop on Personal Sensor for Remote Health Care Monitoring, 2021, Ljubljana, Slovenia, pp. 128-133

[19] CodeMirror User manual and reference guide, 2022, Available from: https://codemirror.net/

[20] Almende B.V. and Contributors, vis-network (2010-2018), GitHub repository, accessed July 2021, https://github.com/visjs/vis-network

Andrej Trost received his Ph.D. 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-system design for academic and industrial applications.

Manfred Ley received his Dipl.-Ing. degree in 1983 from the Faculty of Electrical Engineering, Technical University Graz. After working as a design and project engineer for several electronics and semiconductor companies he joined Carinthia University of Applied Sciences in 1998 to set-up a microelectronics laboratory. Since then he has been teaching electronics, CAD-tools and integrated circuit design on all study levels and has been involved in many research projects in the field of mixed-signal and digital integrated circuit design.

Andrej Žemva received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana (UL) in 1989, 1993 and 1996, respectively. He is Professor at the Faculty of Electrical Engineering, UL, teaching courses on Digital integrated circuits design, Electronic circuit testing and Linear electronics. His current research interests include digital systems design, logic synthesis and optimization, test pattern generation and fault simulation, electronic system-level verification and real-time image processing