E-Learning – VELS
Specification of the VELS tasks

Project:	E-Learning – VELS
Author(s):	Martin Mosbeck, Hedyeh Kholerdi, Gilbert Markum, LingYin Huang
Reviewer:	Axel Jantsch
Version:	0.6
Date:	November 3, 2018

Contents

Version	Autor	Date	Comment
0.1	Martin Mosbeck	06.10.2016	New document, seperated from VELS Specification
0.2	Hedyeh Kholerdi	07.11.2016	Added Tasks ALU,ROM,RAM
0.3	Gilbert Markum	02.03.2017	Added Tasks SC_CU and register_file
0.4	LingYin Huang	31.07.2017	Added Tasks timingDemo
0.5	Martin Mosbeck	20.09.2017	Added Task block code
0.6	Daniel Hauer	20.09.2017	Added Task cache

1. Specification Overview

This document aims to specify the tasks and the interaction and processes which are involved for the VELS system. Tasks are dynamically generated using a task template and parameters. If a student wants a new task the parameters are randomly chosen and inserted into the template. The general goal is that every student of a unique course year gets a similar, but not identical task.

After submission the student's task solution will be checked via a dynamically generated testbench. Before testing, the entity is compiled to check for syntax errors. The testbench feeds the student's design, the UUT (Unit Under Test) with test vectors and compares the UUT's output to the desired output. To check the behavior of the UUT as best as possible and prevent the student to design a UUT that does only have the right behavior for a limited portion of the test vector space, the test vectors are randomly chosen from the test vector space. Furthermore every separate submission is checked by a different test vectors set, either by permuting, choosing a random portion of the test vector space or by varying the test duration. For the case that the testing of a UUT produces an endless loop, each task is tested with a simulation timeout.

To specify each task the following sequential notation is used:

<Task Name>

Overview	General Overview of the the task, the intent and learning objective.
Description	Which informations the student will get to solve the task.
Creation	How the individual task for the student is created on the server.
Submission	What the student has to submit.
Testing	How the student solution is tested on the server.
Feedback	What feedback the student gets after testing.

A course is composed of a task queue that explicitly orders a selection of the available tasks. Creation and modification of this queue is done by the VELS Web Interface.

All files related to a task are located in a directory. This directory contains:

Minimal	• **Generator executable:** Generates random parameters for the task, creates entity and behavioral vhdl files and description test and pdf file for the student. Stores the individual task parameters in the *UserTasks* database table. • **Tester executable:** Given the individual task parameters, tries to compile the student's solution. Generates an individual test bench for the student's solution. Tests the solution and creates feedback text and files for the student.
Optional	• **Scripts:** Scripts that are called from the executables in order to aid the generation or test process • **Templates:** These files have to be filled with parameters and can be used to generate entity declarations, test benches or description texts and files for the students. • **Static:** These files are static for the task, therefore the same for every student. • **Exam:** Files that are needed for the VELS exam mode.

The following sections define a set of initial tasks for the VELS E-learning system. New tasks can be added at any time by a course operator via VELS direct server access.

2. Truth Table

Overview	The student has to program the behavior of an entity which behaves according to a given truth table. The student learns how computations on inputs can be done and and the proper output can be set.
Description	The student receives pdf file with a truth table for 4 inputs and 1 output. It is noted that the task can be solved via look up table or prior simplification via KV (Karnaugh Veith) algorithm. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file with for the student's behavioral implementation.
Creation	In a 4 input system the truth table is composed of $N = 2^4$ rows. Therefore $2^N = 2^{16} = 65536$ possible truth tables can be created. In order to allow KV optimizations as a DNF(Disjunctive Normal Form) the truth table is generated and then checked if optimization is possible using the Quine–McCluskey algorithm.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A testbench tests the students entity with all 16 possible input combinations and compares them to the given truth table via a lookup table.
Feedback	If the syntax analysis returned errors, they are sent to the student. If one of the tests results in a wrong result, the chosen operands and the proper result is sent to the student. The student is informed if his solution was proper or not.

3. Gates

Overview	The student has to program the behavior of an entity which behaves according to a given gate network. The student learns how to use predefined components and connect them.
Description	The student receives a pdf file with a gate network as a picture. It is noted that the tasks should be solved using IEEE 1164 gates. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file with for the behavioral implementation. The to be used IEEE 1164 gates are supplied in the form of the following files: a file for the entities, a file for the behavior and a package file.
Creation	The randomly generated gate network has the following properties: • 2 level depth • maximum of 4 gates per level • 4 inputs, 1 output • gate types: AND, NAND, OR, NOR, XOR, XNOR • every input can be negated The generated network is converted to a pdf file using LaTeX.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A testbench tests the students entity with all 16 possible input combinations and compares the output them to server side calculated proper values.
Feedback	If the syntax analysis returned errors, they are sent to the student. If one of the tests results in a wrong result, the chosen operands and the proper result is sent to the student. The student is informed if his solution was proper or not.

4. PWM(Pulse-Width Modulation) Generator

Overview	The student has to program the behavior of an entity which produces a PWM signal with a given output frequency and a given duty cycle. The output PWM signal shall be generated with the help of a given input clock signal at 50 MHz. The student learns how to handle clock signals for output signal generation.
Description	The student receives information about what PWM is, the frequency of the input clock (50 MHz), the frequency of the desired PWM output signal and the desired duty cycle. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file with for the behavioral implementation. The student is informed that the use of the keywords ’after’ and ’wait’ is prohibited for this task.
Creation	Both the output frequency and duty cycle are randomly generated. The input clock is fixed at 50 MHz.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench feeds the student’s entity with the specified input frequency, tries to calculate a frequency and a duty cycle of the output signal and compares it to the specified frequency and duty cycle. Students’ entities which use the keywords ’after’ and ’wait’ are rejected.
Feedback	If the syntax analysis returned errors, they are sent to the student. The students’ PWM signal is tested for the specified frequency and duty cycle. If the solution is not proper, the student is informed of the measured values and receives a gtkwave file containing his output signal. If no continuous signal is detectable, the student is told so and also receives the gtkwave file. The student is informed if his solution was proper or not.

5. Arithmetic

Overview	The student has to program the behavior of an entity which acts as an adder or subtracter for 2 operands of a given bit width and a given number representation. The student learns how simple add and subtract operations can be implemented in hardware via vhdl
Description	The student receives which type of operation for which operand bit length and number representation has to be implemented. The appropriate flags (carry, overflow) for the given operation are described. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file the behavioral implementation.
Creation	The operation, the bit width of the 2 operands and number representation is randomly chosen from the following choices: • addition or subtraction • two's complement or unsigned • both operands have the same length between 4 and 16 bit The appropriate flags are included as outputs of the predefined entity.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench tests the student's entity with random operands. All error cases (overflow, carry) are at least tested once.
Feedback	If the syntax analysis returned errors, they are sent to the student. If one of the tests results in a wrong result or the appropriate flags were not set, the chosen operands and the proper result including flags is sent to the student. The student is informed if his solution was proper or not.

6. FSM(Finite State Machine) Mealy/Moore

Overview	The student has to program the behavior of an entity which acts as a Mealy/Moore Finite State Machine according to a given state diagram. The student learns how a simple Mealy/Moore Finite State Machine can be implemented with vhdl using synchronous design principles.
Description	The student receives information about Finite State Machines and synchronous design and a pdf file with a picture of a state diagram. It is noted that the whole design should be programmed according to synchronous design. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file with the states predefined for the behavioral implementation.
Creation	The randomly generated state diagram has the following properties: • 5 states • 2 input bits, 2 output bits For testing the state the state machine is in is also a entity output. The generated state diagram is converted to a pdf file via LaTeX.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench feeds the student's entity with inputs and checks both output and state transitions. The state machine is tested extensively to achieve 100% test coverage.
Feedback	If the syntax analysis returned errors, they are sent to the student. If the end state was not reached the student receives information about the last valid state and the next expected state. The student is informed if his solution was proper or not.

7. CRC(Cyclic Redundancy Check Generator)

Overview	The student has to program the behavior of an entity which generates the CRC for data of a given bit width with a given generator polynomial. The student learns how shift register can be used for calculations.
Description	The student receives an explanation about CRC a bit width, a generator polynomial and an example input and result. It is noted that shift registers should be used for implementation. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file the behavioral implementation.
Creation	The generator polynom and the message bit length is randomly generated. The message bit length is chosen between 12 and 24 bit, the generator polynomial degree is chosen between 5 and 11. It is assured that the degree of the generator polynomial is smaller than the message bit length.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench feeds the student's entity sequentially with multiple data words and checks the resulting CRC values.
Feedback	If the syntax analysis returned errors, they are sent to the student. If one of the tests results in a wrong CRC value, the chosen data, the wrong CRC and the proper solution are sent to the student. The student is informed if his solution was proper or not.

8. ALU

Overview	The student has to implement the behavior of ALU based on the instructions which are randomly chosen. The student learns how to prepare VHDL pre-defined components and also learns how to use inputs and outputs.
Description	The student receives a PDF file which describes the instructions and the chosen flag for each one. She has to write the code in a VHDL file with the entity declaration, and she should not change it.
Creation	Tasks are parametrized by the following parameters: • 4 instructions out of {ADD, SUB, AND, OR, XOR, Shift Left (SHL), Shift Right (SHR), Rotate Left (ROL), Rotate Right (ROR), comparator} • One instruction among ADD and SUB • Two instructions among AND, OR and XOR and comparator • One instruction among SHL, SHR, ROL and ROR • One flag out of {overflow, zero, sign, carry, odd parity} • one flag out of {overflow, zero, sign, carry} for ADD or SUB • one flag out of {zero, sign, odd parity} for AND, OR or XOR • the flag is updated for SHL, SHR, ROL, ROR and comparator during the operation • Rising or falling edge for the clock signal The length of data is 4 bits for all students.
Submission	The student has to submit her behavioral implementation VHDL file.
Testing	A test-bench is generated based on generated parameters. The correct output is generated by Python and is added to the VHDL test bench. Each instruction is checked, and the result from the student's code is compared with the correct result. Then, clock and enable signals are checked to work properly.

Feedback	Student's code is analyzed to see if there is any syntax error. The error is sent to the student by email. It is also simulated to be checked if it works properly. For each instruction, the first wrong error is reported: the wrong and expected output and correspondent input is sent to the student. It is said the problem may be due to the incorret implementation or incorrectly using the signals and variables. Then, the student is notified if there is an error in using the clock signal. Finally, she receives an email if there is an error in using the enable signal.

9. ROM

Overview	The student learns how to define arrays and allocate a value to each component of that array. He also learns to define the length of inputs.
Description	The student receives a PDF file with a specification of the ROM. He also receives a VHDL code in which he should implement the behavior of ROM. He also fills the ROM with the values which are randomly generated. He should be careful not to change the input and output deceleration and not to add unnecessary letters or space. ROM needs to be filled with the specified data. A specific amount of ROM size is filled with randomly generated data in a binary representation. The student should write them in his code as the data existing in the ROM. The rest of location is filled with zero in a binary format.
Creation	ROM has the following randomly generated parameters: • Clock cycle: falling or rising edge • Length of address (between 8 to 16 bits) • Length of instruction (between 8 to 16 bits) • Number of data existing in the ROM (between 12 to 20 bits) • Start location of existing instructions (between 50 to 200) • Instructions which exist in the ROM are different from one student to another
Submission	The student has to submit his behavioral implementation VHDL file.
Testing	A test bench is generated based on the parameters which have been chosen for the student. First, the content of each address is checked in the test-bench randomly to see whether the student correctly filled the ROM with the specified data or not. Then, the clock cycle and enable signals are checked to work correctly.
Feedback	If the syntax analysis returns errors, they are sent to the student. If the VHDL simulation does not work properly, the wrong and expected outputs and correspondent inputs are sent to the student. If the clock signal is defined wrong, it is notified to him.

10. RAM

Overview	The student has to program a RAM with a specified size, length of instruction and address and also the number of reading and writing operations at the same time.
Description	The RAM has two address lines, one or two input data, and one or two output data lines and also read/write enables. The ports of the RAM are provided in VHDL code. Based on the number of reading and writing actions, there are four states, one of which is randomly selected for the student. The chosen state has a few behaviors. The student should implement the RAM based on these behaviors.
Creation	Parameters: • Length of each memory cell (an even number between 8 to 20) • Length of address (6 to 12-bit length) • Number of reading and writing actions: • one reading and two writing operations • two reading and one writing operations • two reading and two writing operations • one reading and one writing operations at the same time (even to the same address) • Length of input: word or double word • Length of output: word or double word The result should be produced on the rising edge of the clock signal for all students.
Submission	The student has to submit his behavioral implementation VHDL file.
Testing	Four test benches are provided based on those four states. In other words, each test bench is specific to the number of reading and writing operations. 16 cells of the memory are randomly selected during the test bench generation, and two sets of 16 data are also randomly generated. Then, the related test bench is generated. The test bench first checks the writing process for each address in RAM and then reading process for each address. The tech bech inclues several parts to test each behavior specified in the description file.

Feedback	If the syntax analysis returns errors, they are sent to the student. If the written value cannot be fetched in a reading operation, an error message is sent to the student which tells him that there is an error in writing or reading operation to a specific location. If two reading or two writing processes to different addresses is specified for the student, the result of testing it is sent to the student as well. The result of testing other specifications like ‘double-word’ also is sent.

11. SC_CU(Single Cycle Control Unit)

Overview	The student has to program the behavior of an entity which shall function as a control unit for a single cycle CPU. The student learns how the inner parts of a single cycle CPU work together.
Description	The student receives a pdf file with the datapath visualization for the given single cycle CPU. In the pdf file the student is informed which instructions shall be implemented and if necessary how the ALU is controlled. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file for the behavioral implementation.
Creation	Four instructions are chosen from a pool of 12 instructions (12 choose 4 = 495). The instructions consist of: • Seven R-type instructions: add, sub, AND, OR, XOR, NOR, slt. • Four I-type instructions: lw, sw, beq, bne. • One J-type instruction: j - jump to an address.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench tests the student's entity with the chosen instructions. Conditional instructions (beq, bne) are tested for all conditions.
Feedback	If the syntax analysis returned errors, they are sent to the student. If the student's control signals for an instruction do not achieve what is asked for, then the name of the instruction is sent to the student. The student is informed if his solution was proper or not.

12. Register File

Overview	The student has to program the behavior of an entity which acts as a register file with one special register. The special register acts as a status register and therefore bits can be accessed separately. The student learns how to use arrays of std_logic_vectors and he has to understand signal delay to handle simultaneous read and write situations.
Description	The student receives a pdf with a memory map diagram of the register file he should implement. The bit length and number of the registers is specified in the pdf and whether in a simultaneous read and write situation the input data shall be passed through immediately or at the next rising edge of the input CLK signal. The student receives a vhdl file with the entity declaration, which should not be changed and a vhdl file for the behavioral implementation.
Creation	The width of the registers is chosen from 8, 16 and 32 bit. The number of registers is chosen within the range of 4 to 8 registers. The size of the special register is selected from within 4 to 8 bits. The handling of simultaneous read and write situations in the special register and the remaining registers is chosen separately between immediately bypassing the input value or at the next rising edge of the input CLK signal.
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench writes unique bit strings to the register file while write access is enabled and while write access is disabled and while performing a simultaneous read from the same register and another final read from all registers to check if the data is persistent.
Feedback	If the syntax analysis returned errors, they are sent to the student. If the solution is not proper, the student is informed whether data did not match, was written although write was not enabled, simultaneous read and write was not handled correctly or the data was not persistent. The student is informed if his solution is proper or not.

13. Timing Demo

Overview	The student learns and analyses the timing behavior of signals in hardware programme. He or she has to implement the behavior of signals according to a provided waveform. Understanding difference from signals delay and variable assigned is the aim of this task.
Description	The student receives a pdf which breaks into an example demo and the task description. The timing example gives an instruction and an thorough knowledge of timing analysis in hardware programme using ModelSIM simulator. On the other hand, the second section "Timing Demo Task" gives a simulated waveform diagram of 4 signals output,N ,O ,P and E, up to 300 ns approximately. The constraints for the signals, the assignment of a variable and how two processes are triggered are specified in the pdf. The pupil also receives a vhdl file for example, a vhdl file with the entity declaration, which should not be changed and a vhdl file for the behavioral implementation.
Creation	The assigned values of variable and signal N are randomly chosen between 20 to 110 in integer so are the delay timing for signal O, E, and P. The coefficient numbers for O, E, and P signal assigned are generated randomly between 1 to 3. The signals and variable has the following relation. A := A + value_a N <= N + value_n O <= c1 * A + c2 * N after delay_o P <= c3 * A - c4 * N after delay_p E <= c5 * O + c6 * E after delay_e ,where A is the variable and c1 c6 are the coefficients. Only value_a is given and the rest of the values has to be decided based on the waveform by the student. The generated signal waveform is converted to a pdf file via LaTeX
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A test bench writes only for a checker based on the timing of given output diagram. No inputs needed.
Feedback	If the syntax analysis returned errors, they are sent to the student. If the value of the simulated outputs is not correct at the specified timimg delay, the student receives an feedback with the expected values for 4 signals and the shown values from his/her behavioral vhdl file at the simulated time. The checker checks the design up to 400 ns. The student is informed if his/her solution is proper or not.

14. Blockcode

Overview	The student has to program the behavior of an entity which acts as an intermediate unit between a data source and a sink. The job of this intermediate unit is to convert given datawords to codewords using (n,k) blockcode encoding. The student learns to program an entity that follows given communication constraints (synchronous, possible backpressure).
Description	The student is given an description of the communication system concerning transmission and encoding. Additionally a example waveform view of a few communication cycles is given as diagram with description for each cycle.
Creation	The encoding is parameterized with the following parameters: • dataword length k: between 5 and 7 • codelength n: between 3 and 8 (depending on k) • random (n-k) chosen parity vectors, with at least 2 entries
Submission	The student has to submit his behavioral implementation vhdl file.
Testing	A testbench feeds the student's entity with simulated communication from a data source and checks the outgoing communication in respect to right encoding and given timing constraints. Every testing also tests if the student implemented the due backpressure needed FIFO by: a) filling the FIFO fully and b) fully emptying the FIFO.
Feedback	If the syntax analysis returned errors, they are sent to the student. If the behavior is wrong, the student gets information about the cycle at which the behavior did not fit, expected outgoing communication and received outgoing communication. Additionally the student gets the signal wave file produced by the simulator to help him debug.

15. Cache

Overview	The student has to implement the reading unit of a direct mapped cache. This task relates to the lecture 'Microcomputer' at 'TU Wien' where basic knowledge about cache is taught. The student needs to understand the concept of a direct mapped cache and learns how to handle arrays of std_logic_vectors and how to address parts of an std_logic_vector.
Description	The student receives a PDF file with a specification of the cache. First a sketch and a provided text explane the basic concept of a direct mapped cache and a cache hit or cache miss. This is followed by a detailed list of requirements, which the student should programm in the caches architecture. As the student should only programm the reading unit, a random cache content is also provided in the description. The main focus of this task lies on the correct handling of the address input. Depending on the given bit widths and cache size the student has to split the address into two parts, one representing the cache index and the other the stored tag.
Creation	Cache has the following randomly generated parameters: • Clock cycle: falling or rising edge • Cache size: Number of stored data entries in the cache (between 5 to 32 entries) • Data length: Length of stored data entries (between 4 to 8 bits) • Tag length: Length of the stored tags in the cache (between 3 to 6 bits) • Address length: This can be calculated from cache size and tag length (between 6 to 11 bits) • Cache content: Random cache data and tags are generated from a random key which is individually created
Submission	The student has to submit his behavioral implementation VHDL file.

Testing	A testbench is generated based on the parameters which have been chosen for the student. The testbench starts by applying a correct address with a cache hit and checks, if the cache outputs the correct data and the correct flag. Then the cache is disabled and it is checked, if the outputs stay disabled while different cach hit addresses are applied. Next the cache is enabled, a cache hit address is applied and the correct behaviour is checked. Finally all possible addresses (2^n... n = length of address) are applied and for each address the correct behaviour is checked (If address is a cache hit, the cache must output the correct data and correct flag, otherwise the cache shall set the data output to high impedance and flag to zero). During all these tests it is always checked, if the outputs do not change before the next working edge (either rising or falling, depending on the chosen parameters).
Feedback	If the syntax analysis returns errors, they are sent to the student. If the VHDL simulation does not work properly, the wrong and expected outputs and corresponding inputs are sent to the student. If the clock signal is defined wrong, the student is also notified.