ICT
Institute of
Computer Technology

E-Learning – VELS
Specification of the 2015 tasks

Project:	E-Learning – VELS
Author(s):	Martin Mosbeck
Reviewer:	Axel Jantsch
Version:	0.1
Date:	October 24, 2016

Contents

Version	Autor	Date	Comment
0.1	Martin Mosbeck	06.10.2016	New document, seperated from VELS Specification

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. Basic 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 • ones' complement, 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. Finite State Machine

Overview	The student has to program the behavior of an entity which acts as a Mealy Finite State Machine according to a given state diagram. The student learns how a simple Mealy 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.