Electronic Design Automation (DEDA)

(Rechnergestützter Entwurf Digital (RED))

Laboratories for Event-Driven Circuit Modeling Using *VHDL*

Martin J. W. Schubert, Electronics Laboratory,
Ostbayerische Technische Hochschule Regensburg,
Regensburg, Bavaria, Germany

Abstract. This is a hands-on training for training VHDL.

1 Introduction

This hands-on training is intended to teach VHDL in parallel to the lessons offered in the author's document Event-Driven Circuit Modeling Using *VHDL*.

VHDL is not case sensitive. Intuitively, this document uses ALL UPPERCASE LETTERS for VHDL keywords and lowercase letters for self-defined identifiers, Whereas some uppercase initials may be used to improve readability, e.g. *DataOutBitWidth*.

The organization of this document is as follows:

Chapter 1 is this introduction,
Chapter 2 introduces free available tools for simulation and synthesis of VHDL models
Chapter 3 defines a structural files system.
Chapter 4 teaches structurally organized modeling
Chapter 5 draws relevant conclusion and
Chapter 6 offers references.

2 Simulate a Synthezise a *VHDL* Model

2.1 VHDL Testbench

Create a library *...Model_Files\VHDL\my_lib_flat* and within it file *tb_counter.vhd* according to listing 2.1.

Listing 2.1: File *tb_counter.vhd*: VHDL testbench to test module *counter*.

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
ENTITY tb_counter IS END ENTITY tb_counter;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ARCHITECTURE beh_tb_counter OF tb_counter IS
  CONSTANT M:NATURAL:=5;
  CONSTANT N:NATURAL:=10;
  SIGNAL count2:NATURAL RANGE 0 TO M;
  SIGNAL count0,count1:NATURAL RANGE 0 TO N;
  SIGNAL reset,clock:std_logic:='0';
  SIGNAL en:std_logic_vector(3 DOWNTO 0);
  --
  COMPONENT counter IS
    GENERIC(period:POSITIVE:=10);
    PORT(reset,clock:IN std_logic;
         eni:IN std_logic;
         eno:BUFFER std_logic;
         count:BUFFER NATURAL RANGE 0 TO period-1
        );
  END COMPONENT counter;
  CONSTANT fclk:REAL:=50.0E6;
  -- configuration specification:
  FOR i_cnt0:counter USE CONFIGURATION WORK.cfg_counter;
BEGIN
  clock <= NOT clock AFTER sec/(2.0*fclk);
  reset <= '0', '1' AFTER 12 ns;
  en(3) <= '1', '0' AFTER 4355 ns, '1' AFTER 4855 ns;
  i_cnt2:counter GENERIC MAP(M) PORT MAP(reset,clock,en(3),en(2),count2);
  i_cnt1:counter GENERIC MAP(N) PORT MAP(reset,clock,en(2),en(1),count1);
  i_cnt0:counter GENERIC MAP(N) PORT MAP(reset,clock,en(1),en(0),count0);
END ARCHITECTURE beh_tb_counter;

-- configuration declaration
LIBRARY WORK; USE WORK.ALL;
CONFIGURATION cfg_tb_counter OF tb_counter IS
  FOR beh_tb_counter
    FOR i_cnt1:counter USE ENTITY WORK.counter(rtl_counter_fsm1_dn); END FOR;
    FOR i_cnt2:counter USE ENTITY WORK.counter(rtl_counter_fsm1_dn); END FOR;
    FOR OTHERS:counter USE CONFIGURATION WORK.cfg_counter; END FOR;
  END FOR;
END CONFIGURATION cfg_tb_counter;
```

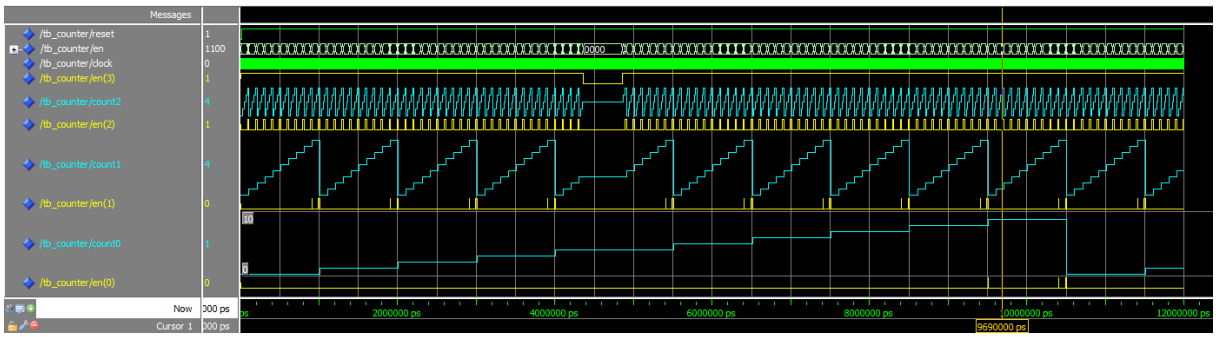

Fig. 2.1: *ModelSim* simulation result of testbench *tb_counter* and *Tcl* files *work.do*, *wave.do*.

2.2 Using the *ModelSim* Simulator

Create files *work.do* and *wave.do* according to listings 2.2.1 and 2.2.2.

Listing 2.2.1: Content of file *work.do* generating Fig. 4.1.

```
vlib work
vmap work work
vcom -work work ./pk_vectors.vhd
vcom -work work ./counter_exercise.vhd
# vcom -work work ./counter.vhd
vcom -work work ./cfg_counter.vhd
vcom -work work ./tb_counter.vhd
vsim work.tb_counter
# vsim work.cfg_tb_counter
do wave.do
run 12 us
```

Listing 2.2.2: Content of file *wave.do* generating the graphics window of Fig. 2.1. (No need to understand this file in detail.)

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /tb_counter/reset
add wave -noupdate -format Literal /tb_counter/en
add wave -noupdate -format Logic /tb_counter/clock
add wave -noupdate -color Yellow -format Logic -itemcolor Yellow /tb_counter/en(3)
add wave -noupdate -color cyan -format Analog-Step -height 36 -itemcolor cyan -offset -0.0 -scale 8.0 /tb_counter/count2
add wave -noupdate -color Yellow -format Logic -itemcolor Yellow /tb_counter/en(2)
add wave -noupdate -color cyan -format Analog-Step -height 74 -itemcolor cyan -offset -0.0 -scale 7.0 /tb_counter/count1
add wave -noupdate -color Yellow -format Logic -itemcolor Yellow /tb_counter/en(1)
add wave -noupdate -color cyan -format Analog-Step -height 74 -itemcolor cyan -offset -0.0 -scale 7.0 /tb_counter/count0
add wave -noupdate -color Yellow -format Logic -itemcolor Yellow /tb_counter/en(0)
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {9690000 ps} 0}
configure wave -namecolwidth 225
configure wave -valuecolwidth 40
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ps} {12600 ns}
```

The *ModelSim* simulator can be obtained free from Intel and is installed in the PC pools of OTH Regensburg.

Clean up directory *my_lib_flat* so that it contains the following files only:

counter_exercise.vhd	entity *counter* and several related empty architectures.
cfg_counter.vhd	configuration for module *counter*.: which architecture will be used?
tb_counter.vhd	testbench containing *tb_counter, beh_tb_counter, cfg_tb_counter*.
work.do	*Tcl* command file generating Fig. 2.1.
wave.do	*Tcl* command file generating a graphics window as e.g. in Fig. 2.1.

Start the *ModelSim* simulator navigate it into your working directory *my_lib_flat*, confirm this directory by listing its contents with command *dir* and run the simulation:

Menu bar: *File → Change Directory → ...\lib_flat → ok*
Transcript window: *dir* (you should see a listing of your files in this directory)
Menu bar: *Tools → Tcl → Execute Macro → work.do*

After starting *work.do* you should see a graphics as shown in Fig. 2.1. However, the counter models are still empty. Alternatively, to the menu bar selections you can type commands into the transcript window, e.g. "*do work.do*" to run the *Tcl* script.

File *work.do* contains *Tcl* commands as detailed in table 2.2. You can type these commands into the transcript window. *Tcl* commands in *ModelSim* are structured as

```
command [-option] argument1 [argument2]
```
A '#' sign in the 1st column of a *Tcl* command line indicates this line as comment.

Table 2.2: Explaining *Tcl* commands.

`# this is a comment line`	# a # in column 1 makes it a comment line
`vlib work`	# Create working library named *work*
`vmap work work`	# Map logical name *work* to working lib. *work*
`vcom -work work counter_empty.vhd`	# Comp. *counter_empty.vhd* to working lib. *work*
`# vcom -work work counter.vhd`	# Compile file *counter.vhd* to working lib. *work*
`vcom -work work cfg_ounter.vhd`	# Compile file *cfg_counter.vhd* to lib. *work*
`vcom -work work tb_counter.vhd`	# Compile file *tb_counter.vhd* to lib. *work*
`vsim work.cfg_tb_counter`	# Simul. config. *cfg_tb_counter* located in *work*
`# vsim work.tb_counter`	# Simul. eitity *tb_counter* located in *work*
`do wave.do`	# Run file *wave.do* defining the *wave*-win
`run 12 us`	# Simulate a time span of 1 µs
`./`	# stands for "this directory"
`../`	# stands for "parent directory"

ModelSim allows us to run and simulate two different top-level modules:
1. `vsim work.tb_counter` # Simulates entity *tb_counter* using its last compiled architecture
2. `vsim work.cfg_tb_counter` # Simulates configuration *cfg_tb_counter* declaring entity + arch.

2.3 Exercises Writing VHDL Code in *ModelSim* Simulator

2.3.1 Getting Started With *ModelSim* and *VHDL*

We will now complete *counter_exercise.vhd* to operate as valid counter.

First, let us configure the situation such, that we can use the models we want to use. In file *work.do,* we find among others the 2 *Tcl* command lines

```
vsim work.cfg_tb_counter
# vsim work.tb_counter
```

The simulator will load the configuration *cfg_tb_counter* as declared at the end of listing 2.1. Here you can select the entity-architecture configurations for the different instantiations. Line

```
FOR OTHERS:counter USE CONFIGURATION WORK.cfg_counter; END FOR;
```

in file *tb_counter.vhd* declares, that the complete component *counter* (consisting of entity and architecture) is embedded within design unit *cfg_counter*.

If we uncomment the second line "`# vsim work.tb_counter`" in file *work.do*, then the simulator will load entity *tb_counter* with its latest compiled architecture, which is *beh_tb_counter*, because it is the only one. Moreover, as there is no configuration now, for all instances of entity counter *ModelSim* will use its latest compiled architecture. This is correct. *Quartus II* will randomly take any architecture for entity counter, so we do not know what happens. Either configure or deliver one architecture only for synthesis with *Quartus II*.

2.3.2 *Counter* with FSM-Loop made of 2 Processes

The following architecture was copied from file *counter_empty.vhd*. Fill the missing code lines below describing the FSM and test the model.

Listing 2.3.2: Architecture *rtl_counter_fsm2_up*.

```
ARCHITECTURE rtl_counter_fsm2_up OF counter IS
  SIGNAL state,nextstate:NATURAL RANGE 0 TO period-1;
BEGIN
  -- Begin NextState Logic:

  p_NextState:PROCESS(.....................)
  BEGIN
.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

  END PROCESS p_NextState; -- End NextState Logic
  --
  -- Begin State Memory
  p_StateMemory:PROCESS(reset)
  BEGIN

  END PROCESS p_StateMemory;
  --
  -- output logic:
                                                    -- Moore outputs
..............................................
                                                    -- Mealy outputs
..............................................
END ARCHITECTURE rtl_counter_fsm2_up;
```

Compile this model with *ModelSim*, either as last architecture in file *counter.vhd* or as own file or in any sequence using the configuration statement. You should see a first success.

Hint: You will find the solution in the author's script ."Event-Driven Circuit Modeling Using VHDL"

2.3.3 *Counter* with FSM-Loop made of 1 Process

The following architecture was copied from file *counter_empty.vhd*. Fill the missing code lines below describing the FSM and test the model.

Listing 2.3.3: Architecture *rtl_counter_fsm1_dn*.

```
ARCHITECTURE rtl_counter_fsm1_dn OF counter IS
  SIGNAL state,nextstate:NATURAL RANGE 0 TO period-1;
BEGIN
  -- Counter
  p_fsm:PROCESS(reset,clock)
  BEGIN

.....................................................

.....................................................

.....................................................

.....................................................

.....................................................

.....................................................

.....................................................

.....................................................

  --
  -- output logic:
                                                  -- Moore outputs
.............................................
                                                  -- Mealy outputs
.............................................
END ARCHITECTURE rtl_counter_fsm1_dn;
```

Compile this model with *ModelSim*, either as last architecture in file *counter.vhd* or as own file or in any sequence using the configuration statement. You should see a first success.

Hint: You will find the solution in the author's script ."Event-Driven Circuit Modeling Using VHDL"

2.4 Synthesize VHDL Model Using *Quartus II*

Start *Quartus II* > *New Project* Wizard > *Next* >
> Working directory: (navigate to directory …*my_lib_flat*\) >
> Name of the project: *counter*
> Name of the top-level design entity: *counter*
> *Next* >*Empty project* > *Next* > *Filename: … (navigate to …\my_lib_flat*\
> *counter* > *Open* > *Next* >
> *Device family* >*Family* > *5CSEMA5F31C6* > *Next* > *Next* > *Finish*
Quartus II Main menu: Processing > *Start Compilation* (= blue arrow button in top bar)
Wait until *Quartus II* finishes Compilation.
Quartus II Main menu: Tools > *Netlist Viewers* > *RTL Viewer* (→ you should get Fig.2.4)

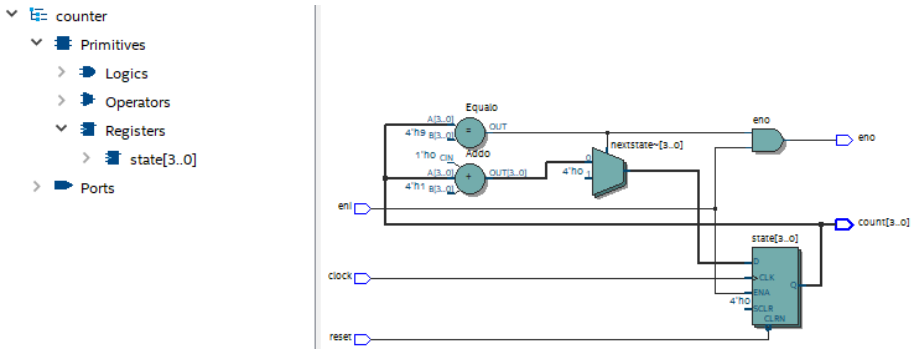

Fig. 2.4: *Quartus II* 18: synthesized model in *RTL Viewer*

2.5 Finishing Work in Flat Directory

Looking in your directory …*my_lib_flat* it becomes obvious that we need an order to not get confused. Clean up directory …*my_lib_flat* so that it contains the following files only:

counter_empty.vhd	entity *counter* and several related empty architectures.
counter.vhd	entity *counter* and several related empty architectures.
cfg_counter.vhd	configuration for module *counter*: which architecture will be used?
tb_counter.vhd	testbench containing *tb_counter, beh_tb_counter, cfg_tb_counter.*
work.do	*Tcl* command file generating Fig. 2.1.
wave.do	*Tcl* command file generating a graphics window as e.g. in Fig. 2.1.

3 Creating a Structural File System

Objective of this chapter is to control, manage and configure a project using different tools.

Seek to maintain *VHDL's* independence for your designs. Dependencies invite to abuse.

Hardware used in the following is the *DE1-SoC* board of *Terasic Inc*. with a *Cyclone V FPGA* from *Intel*.

Software used in the following is the *ModelSim* for Simulation and *Quartus II* for synthesis.

3.1 General Structure

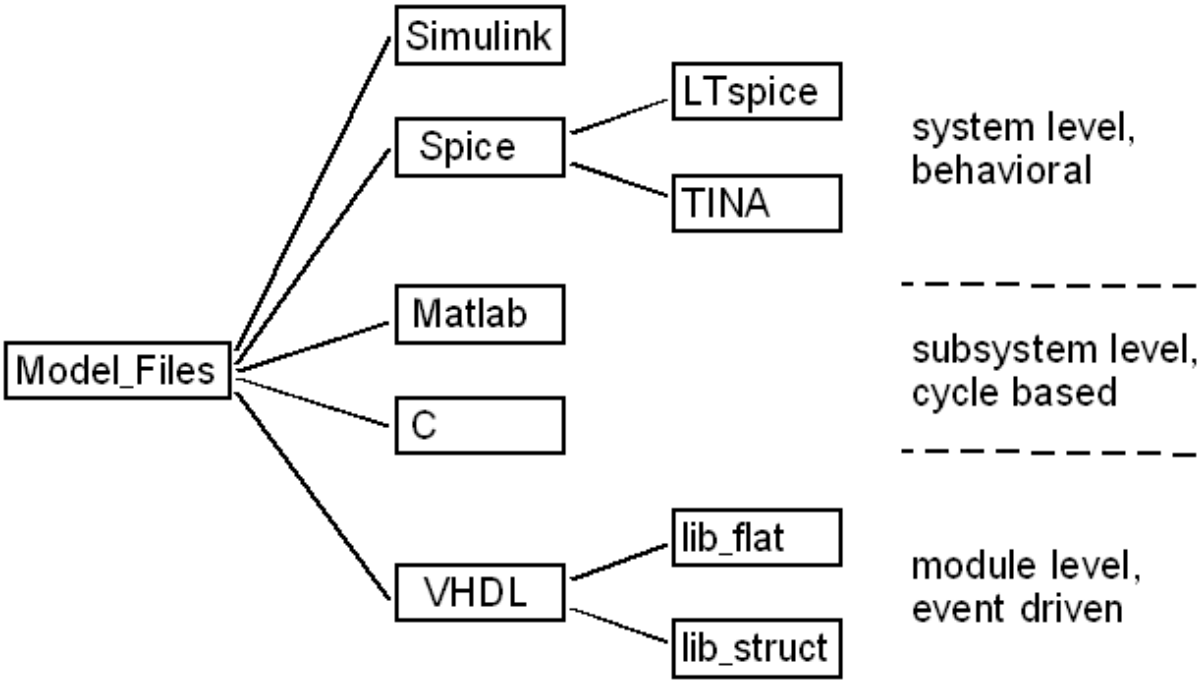

Fig. 3.1: General directory structure for all model files of this course

Fig. 3.1 illustrates the complete file structure proposed by the author.
Fig. 3.2 illustrates the directory structure for all VHDL model files and testbenches.

3.2 *VHDL* Files Structure

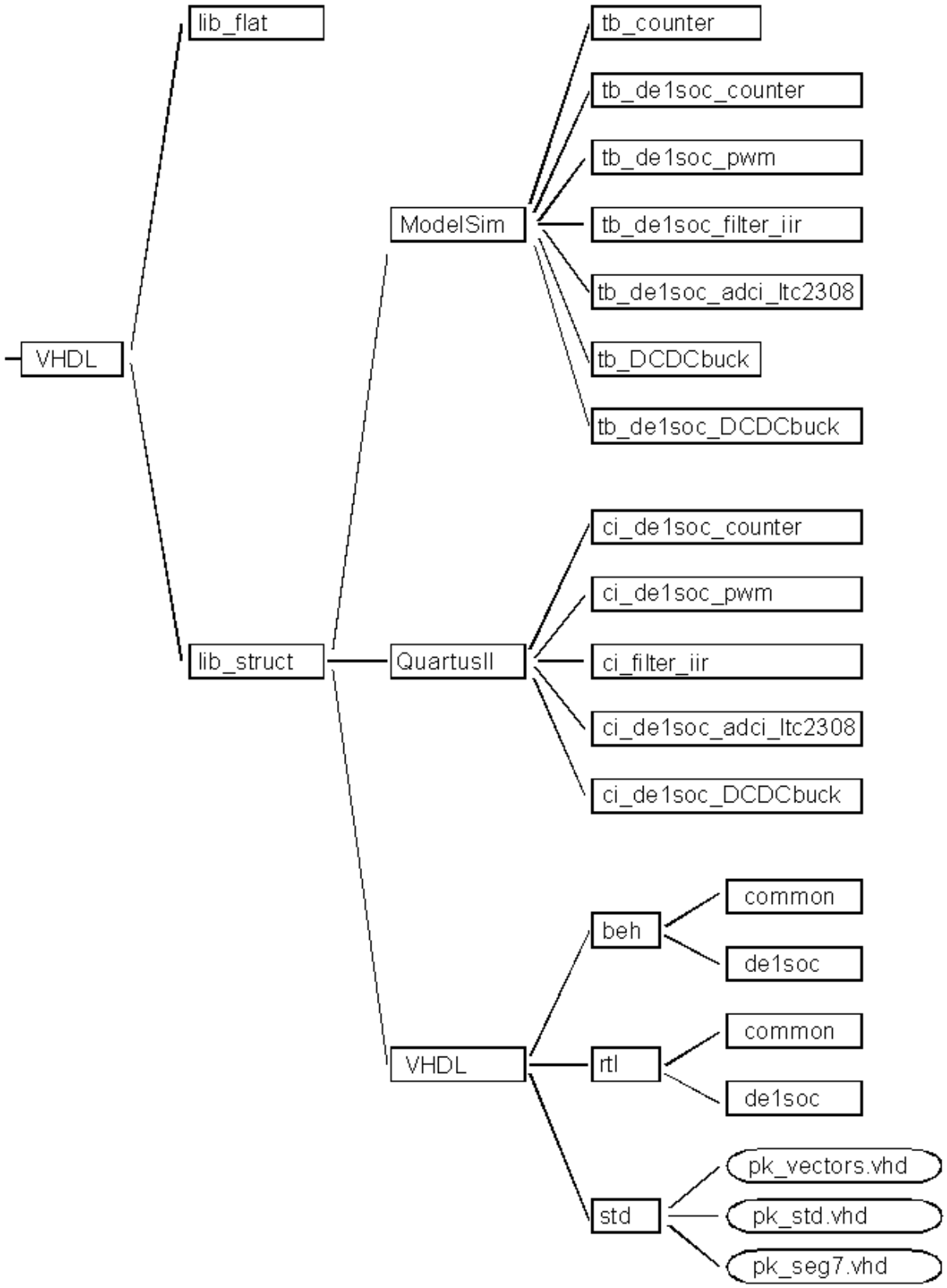

Fig. 3.2: Directory structure for all VHDL files of this course

Significance of directories and files

…\VHDL\lib_flat counter model with several architectures for simulation and synthesis to *DE1-SoC* board.

…\VHDL\lib_struct

\ModelSim: Testbenches

tb_counter:	from *flat* → *struct*: counter model with several architectures
tb_de1soc_counter	from *flat* → *struct*: counter model adopted for *DE1-SoC* board,
tb_de1soc_pwm	counter model extended to pulse-width modulator,
tb_filter_iir	tests different IIR filter models with order $R \leq 2$.
tb_de1soc_adci_ltc2308	testbench for interface to ADC *LTC2308* on *DE1-SoC* board.
tb_DCDCbuck	behavioral model of DC/DC buck converter, clocked with $f_s = 1/T_s$.
tb_de1soc_DCDCbuck	synthesizable model of DC/DC buck converter adopted for *DE1-SoC* board with *DCDCbuck* daughter board.

\QuartusII: Configuration interfaces related to *DE1-SoC* board

ci_de1soc_counter	configuration interface for counter model in *DE1-SoC* board,
ci_de1soc_pwm	configuration interface for pulse-width modulator,
ci_filter_iir	configuration interface to test synthesizable IIR filters, order $R \leq 2$.
ci_de1soc_adci_ltc2308	configuration interface to ADC *LTC2308* on *DE1-SoC* board.
ci_de1soc_DCDCbuck	ci to run *DE1-SOC* board with *DCDCbuck* daughter board.

\VHDL: Tool independent VHDL files

\beh: behavioral (=non-synthesizable) VHDL files

\common: hardware independent VHDL files

adc.vhd	general A/D converter model,
adc_ltc2308.vhd	behavioral model of ADC *LTC2308* on DE1-SoC board,
filterf_canon1.vhd	behavioral model of IIR filter in 1st canonical direct structure using floating point numbers as I/O data,
…	

\rtl: register transfer level (=synthesizable) VHDL files

\common: hardware independent VHDL files

adci.vhd	synthesizable interface to general A/D converter model,
adci_ltc2308.vhd	synthesizable interface to ADC *LTC2308* on DE1-SoC board,
filteri_canon1.vhd	synthesizable model of IIR filter in 1st canonical direct structure using integer type numbers as I/O data,
…	

\de1soc: VHDL interfaces to *DE1-SoC* board

de1soc_counter.vhd	synthesizable interface to fit entity *counter* into *DE1-SOC* board,
…	

\std: standard VHDL packages for this course

pk_vectors.vhd	declaration of data types integer_vector, real_vector,…, that are standard for some VHDL compilers and must not be compiled for them.
pk_std.vhd	declaration of some standard function used I this course.
pk_seg7.vhd	utilities to drive 7-sgment displays, e.g. of *DE1-SoC* board.

Structural Modeling, Simulation and Synthesis

3.3 Getting Started with Structural *counter*

3.3.1 Simulating testbench *tb_counter with* ModelSim

Create the directory *tb_counter* shown in the directory structure of Fig. 3.2. Copy all files of your library *...\my_lib_flat* into the new directory *...\lib_struct\ModelSim\tb_counter*.

Delete file *counter_empty.vhd*.
Move file *counter.vhd* → *...\lib_struct\VHDL\rtl\common\counter\counter.vhd*.
Copy file *cfg_counter.vhd* → *...\lib_struct\VHDL\rtl\common\counter\cfg_counter*.
Remaining files in directory *tb_counter* are *tb_counter, work.do, wave.do.*

Open ModelSim and navigate it to directory *...\lib_struct\ModelSim\tb_counter* as working directory.

Within *ModelSim*: Type *do work.do* into the transcript window. You will get an error message because *ModelSim* does not find files *counter.vhd* and *cfg_counter.vhd* to compile them.

Within *ModelSim*: Type "*edit work.do*" into the transcript window. File *work.do* will open. Click with right mouse button into the editor window and deactivate the *Read Only* flag.
Modify file *work.do* to

Listing 4.1: Content of file *work.do*. in directory *...\lib_struct\ModelSim\tb_counter*.

```
vlib work
vmap work work
vcom -work work ../../VHDL/std/pk_vectors.vhd
vcom -work work ../../VHDL/rtl/common/counter.vhd
vcom -work work ../../VHDL/rtl/de1soc/de1soc_counter.vhd
vcom -work work ../../VHDL/rtl/common/cfg_counter.vhd
#vcom -work work ../../VHDL/rtl/de1soc/cfg_de1soc_counter.vhd
vcom -work work ./tb_de1soc_counter.vhd
vsim -gui  work.tb_de1soc_counter
#vsim -gui  work.cfg_tb_de1soc_counter
do wave.do
run 12 us
```

In listing 4.1 we use relative addressing of files *counter.vhd* and *cfg_counter.vhd* using the Linux commands "**./**" for this directory and "**../**" for parent directory.

In *ModelSim* transcript window type "*do work.do*" to run this file. Yu should now get a graphics similar to Fig. 2.1

Note. We have configuration file *cfg_counter.vhd* within directory *VHDL\rtl\common* and one within the local *\ModelSim\tb_counter*. So you are free to use the standard-configuration in *\VHDL\rtl\common* or a particular one in the local directory *\ModelSim\tb_counter*. The last compiled will override other configurations with same name.

3.4 *de1soc_counter*: Fitting *counter* into the *DE1-SoC* Board

3.4.1 Simulating testbench *tb_de1soc_counter* with *ModelSim*

So far we can simulate the *counter*, but there is no way to synthesize it such, that we can test it. Therefore, we will pack it into a component named *de1soc_counter*, which fits the *counter* module into the environment of the *DE1-SoC* board.

Navigate into directory *...\tb_de1soc_counter*. You will find the files

tb_de1soc_counter.vhd # *VHDL* testbench for interface *de1soc_counter*
work.do # *Tcl* command file for *ModelSim*
wave.do # *Tcl* command file declaring graphics window for *ModelSim*

Start *ModelSim*, navigate it into directory *...\tb_de1soc_counter*. Open file *work.do* and look into it: Now, an additional interface component *de1soc_counter* is compiled. Type *work.do* into the *ModelSim* transcript window. You should get a graphics similar to that in Fig. 2.1.

Delete all new generated file within directory *...\tb_de1soc_counter*.

3.4.2 Synthesis and Download into *DE1-SoC* Board with *Quartus II*

Navigate into directory *...\Model_Files\VHDL\lib_struct\QuartusII\ci_de1soc_counter*. You will find the files.

ci_de1soc_counter.qpf # *Quartus II Project File*
ci_de1soc_counter.vhd # *Quartus II Specification File*
ci_de1soc_counter.vhd # configuration interface for *de1soc_counter.vhd*
output_files\ci_de1soc_counter.cdf # configuration file for the programmer

Double-click left mouse button on file *ci_de1soc_counter.qpf*. *Quartus II 18* will start with the actual directory as working directory. For more detailed handling instructions of *DE1-SoC* board and *Quartus II* see authors document "*Getting Started with DE1-SoC Board*".

- If necessary set programmer: *Assignments > Device > 5CSEMA5F31C6 > OK*.
  *Assignments > Import Assignments > ../../DE1_SoC_pin_assigments.csv > Ope*n *> OK*
- Compile the project: *Processing > Start Compilation*
- Download project into the FPGA: *Tools > Programmer > Start*

Look into file *de1soc_counter*:

- Which key is used as asynchronous global *reset* signal?
- Which key is used as global *enable* signal?
- On which pins can we measure the enable flags *en*(0…8)?
- Observe some *en*(#) flags, #=0…8, with the oscilloscope!

Clean your working directory: remove all files that have not been there before. Exception may be *output_files\ci_de1soc_counter.cdf*, which allows for immediate reprogrmming without new compilation.

3.5 *de1soc_pwm*: Pulse-Width Modulation with *DE1-SoC* Board

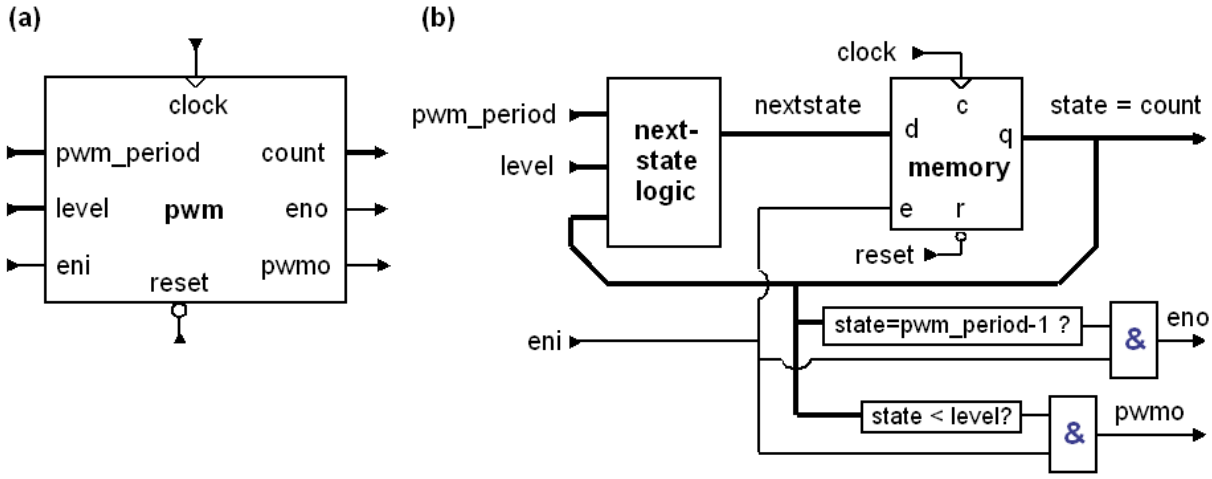

Fig. 4.3: pulse-width modulator: **(a)** symbol, **(b)** schematics: *pwmo*=1 needs *state*<*level* .

A pulse-width modulator is similar to a counter, but has an additional input signal *level* and an additional output signal *pwmo*, whereas

```
pwmo  <= '1' WHEN (state<level AND eni='1') ELSE '0';
```

3.5.1 Simulating Testbench *tb_de1soc_pwm* with *ModelSim*

Navigate into directory *...\tb_de1soc_counter*. You will find the files

tb_de1soc_pwm.vhd	# *VHDL* testbench for interface *de1soc_pwm*
work.do	# *Tcl* command file for *ModelSim*. Change this file such, that # it compiles *pwm_exercise.pwm* instead of *pwm.vhd*.
wave.do	# *Tcl* command file declaring graphics window for *ModelSim*

Within *ModelSim*, complete both architectures in file *pwm_exercise.vhd* such, that they operate as valid pulse-width modulators. If you do not edit file *pwm.vhd* within *ModelSim*, be aware that the file is located at directory *...\VHDL\rtl\common*.

rtl_pwm_fsm1	pulse-width modulator, which counts ascending and realizes the feedback loop of the FSM with a single *VDHL PROCESS*,
rtl_pwm_fsm2	pulse-width modulator, which counts descending and realizes the feedback loop of the FSM with two VDHL PROCESS statements.

After successful simulation, delete all new generated files within directory *...\tb_de1soc_pwm*.

3.5.2 Synthesis and Download of *ci_de1soc_pwm* with *Quartus II*

Navigate to directory *...\Model_Files\VHDL\lib_struct\QuartusII\ci_de1soc_pwm*. You will find the files.

ci_de1soc_pwm.qpf	# *Quartus II Project File*
ci_de1soc_pwm.qsf	# *Quartus II Specification File*
ci_de1soc_pwm.vhd	# configuration interface for *de1soc_pwm.vhd*
output_files\ci_de1soc_pwm.cdf	# configuration file for the programmer

Double-click left mouse button on file *ci_de1soc_pwm.qpf*. *Quartus II 18* will open with the actual directory as working directory. For more detailed handling instructions of *DE1-SoC* board and *Quartus II* see authors document "*Getting Started with DE1-SoC Board*".

- If necessary set programmer: *Assignments > Device > 5CSEMA5F31C6 > OK*.
  *Assignments > Import Assignments > ../../DE1_SoC_pin_assigments.csv > Ope*n > *OK*
- In Quartus II: *Project > Add/Remove Files in Project…*
  remove *../../VHDL/rlt/common/pwm.vhd,* add *pwm_exercise.vhd*.
- Compile the project: *Processing > Start Compilation*
- Download project into the FPGA: *Tools > Programmer > Start*

Look into file *de1soc_pwm*:

- Which key is used as asynchronous global *reset* signal?
- Which key is used as global *enable* signal?
- How do we set the level for module *pmw*?
- What header pins carry signals *pwmo* and *eno*?
- Observe signals *pwmo* and *eno* on the oscilloscope!
- In the configurations specification in module *de1soc_pwm*, combine one after the other architecturs *rtl_pwm_fsm1* and *rtl_pwm_fsm2* for entity *pwm*. Observe the series of 1's on signal *pwmo* relative to flag *eno*. Hint: The configuration specification is made with statement "`FOR i_pwm:pwm USE ENTITY work.pwm(rtl_pwm_fsm1);`"

Close *Quartus II* and clean your working directory: Remove all files that have <u>not</u> been there before. It is up to you if you want to save file be *output_files\ci_de1soc_pwm.sof*, which allows for immediate reprogramming without new compilation.

3.6 Second Order IIR Filter: *f_filteri_canon1_order2*

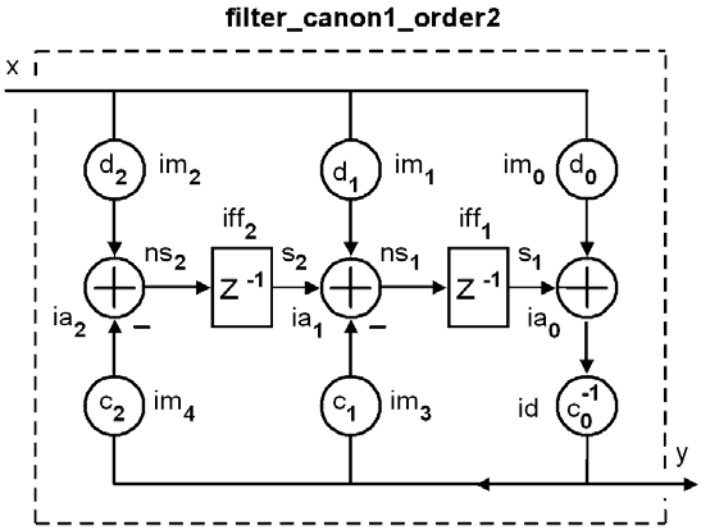

Fig. 4.4: Schematic of module *f_filter_canon1_order2*

Exercise: Assign signal names in Fig. 4.4 to signals of general FSM.

Stimuli:
. . . . . .

Nextstate:
. . . . . .

State:
. . . . . .

Nextstate Logic:
. . . . . .

. . . . . .

Moore Outputs:
. . . . . .

Mealy Outputs:
. . . . . .

Output Logic Moore:
. . . . . .

Output Logic Mealy:
. . . . . .

Solution from document *Matlab_LTI+CacleBased_Modeling_sol.pdf* chapter 4.3.1, page 24:

Stimuli:	`x`
Nextstate:	`ns1, ns2`
State:	`s1, s2`
Nextstate Logic:	`ns1 = d1·x – c1·y + s2`
	`ns2 = d2·x – c2·y`
Moore Outputs:	`<none>`
Mealy Outputs:	`eno, pwmo`
Output Logic Moore:	`<none>`
Output Logic Mealy:	`y <= (d0·x + s1) / c0`

3.6.1 Simulating Testbench *tb_de1soc_filter* with *ModelSim*

Navigate to directory *...\Model_Files\VHDL\lib_struct\ModelSim\tb_filter_iir*. You will find the files

tb_de1soc_filter_iir.vhd	# *VHDL* testbench for interface *de1soc_pwm*
work.do	# *Tcl* command file for *ModelSim*
wave.do	# *Tcl* command file declaring graphics window for *ModelSim*

In work.do change strings "*_canon1_order2*" to "*_canon1_order2_exeercise*". Start *ModelSim*, change its working directory to directory *.../filter_iir*, and run ModelSim with command *work.do*.

Complete the architectures in both files:

...\VHDL\beh\common\filterf_canon1_order2_exercise.vhd	-- 2nd order IIR, float I/O
...\VHDL\rtl\common\filteri_canon1_order2_exercise.vhd	-- 2nd order IIR, integer I/O

The *ModelSim* simulation of *tb_filter_iir* delivers 5 very similar step responses. Your own models should match the given models of arbitrary order *R*.

Delete all new generated file within directory *...\ tb_filter_iir*

3.6.2 Synthesis and Download of *ci_de1soc_filter* with *Quartus II*

Navigate into directory *...\Model_Files\VHDL\lib_struct\QuartusII\ci_de1soc_filter_iir*. You will find the files.

ci_de1soc_pwm.qpf	# *Quartus II Project File*
ci_de1soc_pwm.qsf	# *Quartus II Specification File*
ci_de1soc_pwm.vhd	# configuration interface for *de1soc_filter_iir.vhd*
output_files\ci_de1soc_pwm.cdf	# configuration file for the programmer

Double-click left mouse button on file *ci_de1soc_pwm.qpf*. *Quartus II 18* will open with the actual directory as working directory. Compile and download into the *DE1-SoC* board. This should work without errors. Quit *Quartus II*.

Clean your working directory: remove all files that have not been there before. Exception may be *output_files\ci_de1soc_filter_iir.sof*, which allows for immediate reprogramming without new compilation.

3.7 DC/DC Buck Converter: *DCDCbuck_pcb00*

3.7.1 Simulating Testbench *tb_de1soc_filter* with *ModelSim*

Navigate to directory *...\Model_Files\VHDL\lib_struct\ModelSim\tb_DCDCbuck_pcb00*. You will find the files

tb_de1soc_DCDCbuck.vhd	# *VHDL* testbench for interface *de1soc_pwm*
work.do	# *Tcl* command file for *ModelSim*
wave.do	# *Tcl* command file declaring graphics window for *ModelSim*

In file *work.do,* change all strings "*_canon1_order2*" to "*_canon1_order2_exeercise*" and "*pwm*" to "*pwm_exercise*". Start *ModelSim*, change its working directory to directory *.../tb_de1soc_DCDCbuck* and run ModelSim with command *work.do*.

You should now get a simulation of the DC/DC buck converter for the imaginary board *pcb00.*

Delete all new generated file within directory *...\ tb_DCDCbuck_pcb00*

3.7.2 Synthesis and Download of *ci_de1soc_filter* with *Quartus II*

Navigate to directory *...\Model_Files\VHDL\lib_struct\QuartusII\ tb_DCDCbuck_pcb00*. You will find the files.

ci_de1soc_DCDCbuck.qpf	# *Quartus II Project File*
ci_de1soc_ DCDCbuck.qsf	# *Quartus II Specification File*
ci_de1soc_ DCDCbuck.vhd	# configuration interface for *de1soc_filter_iir.vhd*
output_files\ci_de1soc_ DCDCbuck.cdf	# configuration file for the programmer

Double-click left mouse button on file *ci_de1soc_pwm.qpf*. *Quartus II 18* will open with the actual directory as working directory. Compile and download into the *DE1-SoC* board. This should work without errors. Quit *Quartus II*.

Clean your working directory: remove all files that have not been there before. Exception may be *output_files\ci_de1socDCDCbuck.sof*, which allows for immediate reprogramming without new compilation.

3.8 Model and Optimize Your >Own DC/DC Buck Converter

Copy directories
tb_DCDCbuck_pcb00 → *tb_DCDCbuck_pcb##* with ## being the number of your board.
ci_DCDCbuck_pcb00 → *ci_DCDCbuck_pcb##* with ## being the number of your board.

Replace device parameters of imaginary board *pcb00* with the device parameters that you have characterized for your own board *pcb##*.

Compare simulations of *Simulink*, *LTspice*, *VHDL* and measurements with your board *pcb##*. Measure and optimize the step response of your hardware. Good luck!

4 Conclusions

This tutorial introduces VHDL and applies it to the example of a DC/DC buck converter.

- VHDL is a concurrent, event-driven modeling language.

Fundamental design units are entity, architecture and configuration, whereas the
- Entity corresponds to a symbol, the
- Architecture corresponds to a schematic and the
- Configuration defines entity – architecture combinations, either as
  + configuration declaration, which is a design unit, or as
  + configuration specification, which is a statement in the architecture.
- Packages may have a package body and are useful to declare data types and functions.

Code can be written
- Concurrent, which is the default,
- Sequential, which is within processes and subprograms, and
- Structural, which defines a hierarchy from modules and submodules.

FSM design
The feedback loop of a finite state machine (FSM) consists of combinational nextstate logic and state memory.
- The state memory has to be written as process.
- The nextstate logic must not contain memory and can be written
  + concurrent statements or as
  + process that does not generate memory.
- Furthermore, nextstate logic and state memory can be written as a single process.

Processes run top-down within a single simulation delta (Δ), when an event happens in their sensitivity list. To describe combinational logic without memory with a process,
- All its input signals must be listed in the sensitivity list,
- All its output signal must be driven in any possible situation.

Data:

We have 3 kinds of data objects:
- Signals, which are always assigned with a delay, at least one Δ,
- Variables, that are assigned immediately and exist within sequential environments only,
- Constants, which can be seen as variables during compile time and constants afterwards.

A data object may have any data type. There are 4 scalar data types:
- Real, which are floating point numbers of either 4 or 8 bytes, depending on the compiler,
- Integer, which are integral numbers of either 4 or 8 bytes, depending on the compiler,
- Physical, which are integers with a dimension, only predefined type is time,
- Enumerated, which is e.g. used to build state-value systems such as Boolean.

Furthermore, there are 2 composed data types:
- Array, whose elements are addressed with integer numbers, and
- Record, whose fields are addressed with field names (corresp. to structs in Matlab and C).

5 References

[1] *1076 IEEE Standard VHDL Language Reference Manual*, Revision of IEEE Std. 1076, 2002 Edition.

[2] *OVI Verilog HDL Language Reference Manual, version 1.0*, Open Verilog International, 1991.

[3] Available: http://www.systemc.org/home/

[4] Simulink HDL Coder: Generate HDL code from Simulink models and MATLAB code, available: http://www.mathworks.de/products/slhdlcoder/.

[5] Available: http://model.com/

[6] Altera Corporation, available: URL: www.altera.com

[7] M. Schubert, "VHDL Course", *Electronic Design Automation Course*, Regensburg University of Applied Sciences, available: http://homepages.fh-regensburg.de/~scm39115/ → Offered Education → Courses and Laboratories → RED → VHDL

[8] Lehmann, Wunder, Selz, Schaltungsdesign mit VHDL, Franzis' Verlag, Poing 1994.

[9] M. Schubert, Script "Systemkonzepte", Regensburg University of Applied Sciences, available: http://homepages.fh-regensburg.de/~scm39115/ → Offered Education → Courses and Laboratories → SK

[10] M. Schubert, "FSM Design for DSP Using Fixed-Point Numbers", *Electronic Design Automation Course*, Regensburg University of Applied Sciences, available: http://homepages.fh-regensburg.de/~scm39115/ → Offered Education → Courses and Laboratories → RED

[11] M. Schubert, "FSM Design for DSP Using Matlab", *Electronic Design Automation Course*, Regensburg University of Applied Sciences, available: http://homepages.fh-regensburg.de/~scm39115/ → Offered Education → Courses and Laboratories → RED

[12] Altera DE2 Board, produced by Terasic, available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=30

[13] Available: http://en.wikipedia.org/wiki/Reinventing_the_wheel

[14] Keating, Michael; Bricaud, Pierre, "Reuse methodology manual for System-on-a-Chip Designs", Kluwer Academic Publishers, 1999, ISBN 0-7923-8175-0.

[15] Regensburg University of Applied Sciences, internal network drive k:\Sb\EDA\Altera\Software.