Turun yliopisto
University of Turku

HDL Based Design
ME620138

Passing the Course

- Examination (100 % of the grade)
  - Either in December at Fudan University or in January at University of Turku
- All the exercises accepted by the lecturers
- Possibility to increase grade by doing the 7th exercise

The most important thing is to make time for the exercises

Turun yliopisto
University of Turku

Submitting Exercises

Turun yliopisto
University of Turku

Preparing Your Submission

- After you have finished an exercise, execute the following commands in Modelsim in the directory where you have your source files:

```
.main clear (note the dot in front of main)
vdel -all -lib work
vlib work
transcript file exN.log
vcom -f compile_vhdl.f
vsim -do "run -all" top
quit -sim
transcript file ""
```

N is the number of the exercise

-voptargs=+acc
OR
-novopt

Turun yliopisto
University of Turku

.f

- **.f** is just a text file that has all the files listed in the order in which the source files **must** be compiled. One file per line.

```
lights.vhd
lights_tb.vhd
```

- Compilation order follows bottom-to-top approach
  - If a module has a submodule, the submodule **must** be compiled first

For Each Exercise

- Create a directory named exN
- Copy **ONLY** the following files into the directory:

  source file(s)

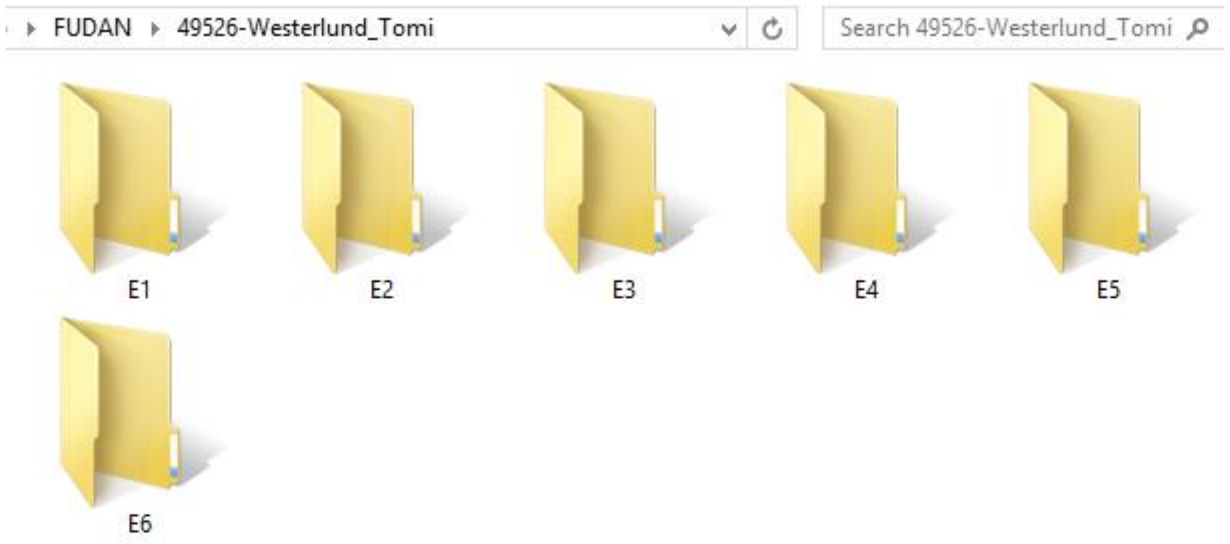
  exN.log

  compile_vhdl.f

Under no circumstances include the work library

Last But One

- Create a directory called
  StudentNumber-Lastname-Firstname
- Within that directory copy your exercises



This is done for all the exercises that you submit.

Finally

- Pack **the directory** using ZIP
- Use, for example, http://www.7-zip.org/
  - Supported formats:
    - Packing / unpacking: 7z, XZ, BZIP2, GZIP, TAR, ZIP and WIM
    - Unpacking only: ARJ, CAB, CHM, CPIO, CramFS, DEB, DMG, FAT, HFS, ISO, LZH, LZMA, MBR, MSI, NSIS, NTFS, RAR, RPM, SquashFS, UDF, VHD, WIM, XAR and Z.
- Filename is the same as the directory name

Check List

- ✓ Have you read and understood what you have to do in the exercise?
  - ✓ **This is important!**
- ✓ Is your exercise complete?
- ✓ Does your system compile after removing the work library?
  - ▪ You must try to compile your exercise after removing the work library
- ✓ **Have you followed the submission steps carefully?**

Turun yliopisto
University of Turku

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com
scottadams@aol.com
11-26-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.

Coding Instructions

Purpose of VHDL Coding Instructions

- Prevent harmful or unpractical ways of coding
- Introduce a common, clear appearance for VHDL
- Increase readability for reviewing purposes
- Not to restrict creativity in any way
- Bad example:

```
A_37894 :process(xR,CK ,datai , DATAO )
BEGIN
if(XR ='1')THEN DATAO<= "1010";end if;
if(CK'event) THEN if CK = '1'THEN
for ARGH in 0
to 3 Loop DATAO(ARGH) <=datai(ARGH);
end Loop;end if;
```

Turun yliopisto
University of Turku

Purpose of VHDL Coding Instructions

```
A_37894 :process(xR,CK ,datai , DATAO )
BEGIN
if(XR ='1')THEN DATAO<= "1010";end if;
if(CK'event) THEN if CK = '1'THEN
for ARGH in 0
to 3 Loop DATAO(ARGH) <=datai(ARGH);
end Loop;end if;
```

File contents and naming

- One VHDL file should contain one entity and one architecture, file named as

```
entityname.vhd
```

- Package name should be

```
packagename_pkg.vhd
```

- Test bench name should be

```
entityname_tb.vhd
```

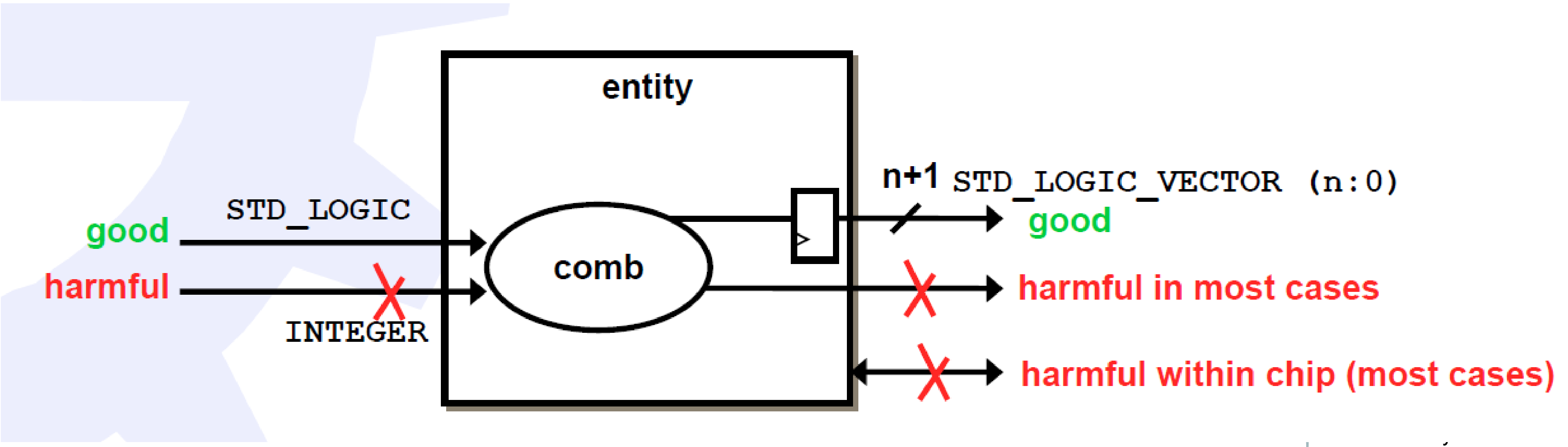
- **A VHDL file and the entity it contains have the same name**

Testbench

- Each entity requires at least one testbench
  - Design without a testbench is useless
- Prefer self-checking testbenches
  - Cannot assume that the verifier looks at the “magic spots” in waveform
  - (Occasionally, TB just generates few inputs to show the intended behaviour in common case)
- Informs whether the test was successful or not
- There can be several test benches for testing different aspects of the design

Entity ports

- Use only modes IN and OUT in the port
  - Signal names have corresponding post-fixes
- Use only signal types STD_LOGIC and STD_LOGIC_VECTOR in the ports
- Output of a block should always come directly from a register



Sequential/synchronous process

- Sensitivity list of a synchronous process has always exactly two signals
  - Clock, rising edge used, named clk
  - Asynchronous reset, active low, named rst_n
- Signals that are assigned inside sync process, will become D-flip flops at synthesis
- Never give initial value to signal at declarative part
  - It is not supported by synthesis (but causes only a warning)

```
SIGNAL main_state_r : state_type := "11110000";
```

- **Assign values for control registers during reset**
- **Synchronous process is sensitive only to reset and clock**

Sequential/synchronous process

- Correct way of defining synchronous process:

```
cmd_register : PROCESS (rst_n, clk)
BEGIN
   IF (rst_n = '0') THEN
      cmd_r <= (OTHERS => '0');
   ELSIF (clk'EVENT AND clk = '1') THEN
      cmd_r <= …;
   END IF;
END PROCESS cmd_register;
```

- **Clock event is always to the rising edge**
- **Assign values to control registers during reset**

Combinatorial/asynchronous process

- An asynchronous process must have **all input signals in the sensitivity list**
  - If not, simulation is not correct
  - Top-3 mistake in VHDL
  - Input signals are on the right side of assignment or in conditions (if, for)
- If-clauses must be complete
  - Cover all cases, e.g. with elsebranch
  - All signals assigned in every branch
  - Otherwise, you’ll get latches (which are evil)

Combinatorial/asynchronous process

- An example of an asynchronous process:

```
decode : PROCESS (cmd_r, bit_in, enable_in)
BEGIN
   IF (cmd_r = match_bits_c) THEN
      match_0       <= '1';
      IF (bit_in(1) = ‘1’and bit_in(0) = ‘0’) THEN
         match_both <= enable_in;
      ELSE
         match_both <= '0';
      END IF;
   ELSE --else branch needed to avoid latches
      match_0       <= '0';
      match_both    <= '0';
   END IF;
END PROCESS decode;
```

- Same signal cannot be on both sides of assignment in combinatorial process
  - That would create combinatorial loop, i.e. malfunction

Turun yliopisto
University of Turku

Naming Conventions

▪ General register output	`signalname_r`
▪ Combinatorial signal	`signalname`
▪ Signal Between components	`signalname_a_b`
▪ To multiple components	`signalname_from_a`
▪ Input port	`portname_in`
▪ Output port	`portname_out`
▪ Constant	`constantname_c`
▪ Generic	`genericname_g`
▪ Variable	`variablename_v`

Important is that the signal name clearly indicates its source and usage

Clk and reset signals/inputs

- Active low reset is `rst_n`
  - Asynchronous set should not be used
- Clock signal `clk`
  - If there are more clocks the letters "clk“ appear in every one as a postfix

- When a signal ascends through hierarchy, its name should remain the same. This is especially important for clock signal

Naming in general

- Descriptive, unambiguous names are very important
- Names are derived from English language
- Use only characters
  - alphabets ‘A’.. ‘Z’,‘a' .. ‘z',
  - numbers '0' .. '9‘ and underscore'_'.
  - First letter must be an alphabet
- Use enumeration for coding states in FSM
  - Do not use: s0, s1, a, b, state0, ...
  - Use: idle, wait_for_x, start_tx, read_mem, ...

- Average length of a good name is 8 to 16 characters

Signal types

- Direction of bits in

  `STD_LOGIC_VECTOR` **is always** `DOWNTO`

- Size of the vector should be parameterized
- Usually the least significant bit is numbered as zero (not one!):

```
SIGNAL data_r :
STD_LOGIC_VECTOR(datawidth_g-1 DOWNTO 0);
```

- Use package numeric_std for arithmetic operations

Named signal mapping in instantiations

- Recommended to use named signal mapping, not ordered mapping

```
i_datamux : datamux
 PORT MAP (
    sel_in   => sel_ctrl_datamux,
    data_in  => data_ctrl_datamux,
    data_out => data_datamux_alu
    );
```

- This mapping works even if the declaration order of ports in entity changes

Use assertions

- Easier to find error location
- Checking always on, not just in testbench
- Assertions are not regarded by synthesis
- tools -> no extra logic

```
assert (we_in and re_in)=0
report ”Two enable signals must not active
at the same time”
severity warning;
```

- If condition is not true during simulation,
  - the report text, time stamp, component where it happened will be printed
- Ensure that your initial assumptions hold
  - e.g. data_width is multiple of 8 (bits)

Comment thoroughly

- Comment the intended function
  - Especially the **purpose** of signals
  - Not the VHDL syntax or semantics
  - Think of yourself reading the code after a decade
- A comment is indented like regular code
  - A comment is placed with the part of code to be commented.
- Be sure to update the comments if the code changes
  - Erroneous comment is more harmful than not having a comment at all

Code appearance: Aligning

- Align the colons and port types in the entity port:

```
ENTITY transmogrifier IS
   PORT (
      rst_n      : IN STD_LOGIC;
      clk        : IN STD_LOGIC;
      we_in      : IN  STD_LOGIC;
      cmd_0_in   : IN  STD_LOGIC_VECTOR(3-1 DOWNTO 0);
      data_in    : IN  STD_LOGIC_VECTOR(5-1 DOWNTO 0);
      valid_out  : OUT STD_LOGIC;
      result_out : OUT STD_LOGIC_VECTOR(6-1 DOWNTO 0)
);
END transmogrifier;
```

Code appearance: Aligning

- Align colons inside one signal declaration group:

```
--control signals
SIGNAL select     : STD_LOGIC_VECTOR (2-1 DOWNTO 0);
SIGNAL cmd_r      : STD_LOGIC_VECTOR (32-1 DOWNTO 0);
SIGNAL next_state : state_type;
```

```
--address and data signals
SIGNAL rd_addr  : STD_LOGIC_VECTOR (16-1 DOWNTO 0);
SIGNAL wr_addr  : STD_LOGIC_VECTOR (16-1 DOWNTO 0);
SIGNAL rd_data  : STD_LOGIC_VECTOR (32-1 DOWNTO 0);
```

Code appearance: Aligning

- Align the => in port maps:

```
i_pokerhand : pokerhand
PORT MAP (
   rst_n      => rst_n,
   clk        => clk,
   card_0_in => card (i),
   card_1_in => card (i),
   card_2_in => card (i),
   card_3_in => card (i),
   card_4_in => card (i),
   hand_out  => hand
   );
```

Coding Instructions

1. Entity ports
   - Use only modes IN and OUT with names having suffixes _in or _out
2. Only types STD_LOGIC and STD_LOGIC_VECTOR
3. Use registered outputs
4. A VHDL file and the entity it contains have the same name
5. One entity+architecture per file
   - Every entity has a testbench
6. Synchronous process
   - always sensitive only to reset and clock
   - clock event is always to the rising edge
   - all control registers must be initialized in reset

Coding Instructions

7. Combinatorial process's sensitivity list includes all signals that are read in the process
   - Complete if-clauses must be used. Signals are assigned in every branch.
8. Use signal naming conventions
9. Indexes of STD_LOGIC_VECTOR are defined as DOWNTO
10. Use named signal mapping in component instantiation, never ordered mapping
11. Use assertions
12. Write enough comments

Turun yliopisto
University of Turku

Coding Instructions

13. Every VHDL file starts with a header
14. Indent the code, keep lines shorter than 76 characters
15. Use descriptive names
16. Label every process and generate-clause
17. Clock is named clk and async. active-low reset rst_n
18. Intermediate signals define source and destination blocks
19. Instance is named according to entity name
20. Declare signals in consistent groups

Turun yliopisto
University of Turku

Some Extra Reading Material

Timing

Files

DELAY MECHANISMS

Delay Mechanisms

- Transport and Inertial delays

  target <= [*delay_mechanism*] waveform

  delay_mechanism <= **transport** I [**reject** time_expression] **inertial**

- Transport delay mechanism is used when we model an ideal device
  - An output follows all the changes in an input

- Inertial delay mechanism is used when we model a real device
  - An output follows input that are applied for a sufficiently long duration
  - Inertial delay is used when a signal assingment does not have a delay

Delay Mechanisms

```
Dout <= transport Din after 10 ns;
```

- The following assingments are equivalent

```
Dout <= Din after 10 ns;
Dout <= inertial Din after 10 ns;
Dout <= reject 10 ns inertial Din after 10 ns;
```

Cannot be greater than the speficied delay

- Dropping the pulse rejection limit is the same as setting the limit equal to the specified delay, and setting the limit to 0 is the same as specifying transport delay

Turun yliopisto
University of Turku

FILES

Files

- Basic interface between VHDL models and a host environment
- File input and output cannot be synthesised
- Files are used to store data that will be loaded into a model, or stored the results produced by a simulation
- Files can only contain one type of object
  - type can be almost any VHDL type
- Used mostly in Testbenches
- Only sequential access to files using commands:
  - open, close, read and write

Opening

- Once we have defined a file

```
type intfile is file of integer; -- Numbers
```

- We can declare file objects

```
file integer_file: intfile open write_mode is “intfile.txt”;
```

open is default so

```
file integer_file: textfile is “intfile.txt”;
```

is equivalent with it

- Text file is predefine. Simply use *text* instead of your own type

Turun yliopisto
University of Turku

Accessing

- Input means std_input and output is std_output

```
file input : text open read_mode is “std_input”;
file output : text open write_mode is “std_output”;
```

- Being able to read from or write into a file, one need to describe a LINE variable

```
variable buf_out, buf_in: LINE;
```

- Using the following command, data is taken out or put into the LINE variable

```
procedure readline(file F: text; L: inout line);
procedure writeline ( file F : text; L : inout line );
procedure readline(file F: text; L: inout line);

procedure read ( L : inout line;  value: out bit;
                                  good : out boolean );
procedure read ( L : inout line;  value: out bit );

procedure read ( L : inout line;  value: out bit_vector;
                                  good : out boolean );
procedure read ( L : inout line;  value: out bit_vector );

procedure read ( L : inout line;  value: out boolean;
                                  good : out boolean );
procedure read ( L : inout line;  value: out boolean );

procedure read ( L : inout line;  value: out character;
                                  good : out boolean );
procedure read ( L : inout line;  value: out character );

procedure read ( L : inout line;  value: out integer;
                                  good : out boolean );
procedure read ( L : inout line;  value: out integer );

procedure read ( L : inout line;  value: out real;
                                  good : out boolean );
procedure read ( L : inout line;  value: out real );

procedure read ( L : inout line;  value: out string;
                                  good : out boolean );
procedure read ( L : inout line;  value: out string );

procedure read ( L : inout line;  value: out time;
                                  good : out boolean );
procedure read ( L : inout line;  value: out time );
```

```
procedure writeline ( file F : text;  L : inout line );

procedure tee ( file F: text;  L: inout line );

function justify ( value: string;
                   justified: side := right;
                   field: width := 0 ) return string;

procedure write ( L : inout line;  value : in bit;
                 justified: in side := right;
                 field: in width := 0 );

procedure write ( L : inout line;  value : in bit_vector;
                 justified: in side := right;
                 field: in width := 0 );

procedure write ( L : inout line;  value : in boolean;
                 justified: in side := right;
                 field: in width := 0 );

procedure write ( L : inout line;  value : in character;
                 justified: in side := right;
                 field: in width := 0 );

procedure write ( L : inout line;  value : in integer;
                 justified: in side := right;
                 field: in width := 0 );

procedure write ( L : inout line;  value : in real;
                 justified: in side := right;
                 field: in width := 0;
                 digits: in natural := 0 );

procedure write ( L: inout line;  value: in real;
                 format: in string);

procedure write ( L : inout line;  value : in string;
                 justified: in side := right;
                 field: in width := 0 );
```

```
--writing to standard output
write(buf_out, string'("Enter the parameter"));
writeline(output, buf_out);

--reading user's input from standard input
readline(input, buf_in);

--reading the value from the line variable to a integer variable
read(buf_in, count);

--printing the written value first to a line variable and then to
--  standard output and finally to an integer file.
write(buf_out, string'("The parameter is="));
write(buf_out, count);
writeline(output, buf_out);
```

```
while not endfile(infile) loop
        -- printing to standard output
        write(buf_out, string'("The parameter from an input file is="));
        writeline(output, buf_out);
        -- reading from in input file
        readline(infile, buf_in);
        read(buf_in, count);
        -- storing the data for future use, etc.
        writeline(output, buf_in);
end loop;
```