PlTbUtils Specification

Author: Per Larsson
pela@opencores.org

Rev. 0.4
January 09, 2014

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	9/2/2013	Per Larsson	First draft
0.2	11/10/2013	Per Larsson	Added sections Acknowledgements and Language. Added reference section on waitsig(). Updated reference section on print() and pltbutils_clkgen.
0.3	1/5/2013	Per Larsson	Added sections User Configuration, Configuring Simulation Halt, Configuring Messages for Integration Environments. In reference section added starttest, endtest, removed testname. Updated figures and feature bullets.
0.4	1/9/2013	Per Larsson	Updates for alpha0006: Text modified in numerous places to reflect that PlTbUtils is now using the variable pltbv and the signal pltbs for control and status, instead of the previous shared variable and global signals.

1

Introduction

Overview

PlTbUtils makes it easy to create automatic, self-checking simulation testbenches, and to locate bugs during a simulation. It is a collection of functions, procedures and testbench components that simplifies creation of stimuli and checking results of a device under test.

Features:

- Simulation status printed in transcript windows as well as in waveform window (error count, checks count, number and name of current test, etc).
- Check procedures which output meaningful information when a check fails.
- Clear SUCCESS/FAIL message at end of simulation.
- Easy to locate point in time of bugs, by studying increments of the error counter in the waveform window.
- User-defined information messages in the waveformwindow about what is currently going on.
- Transcript outputs prepared for parsing by scripts, e.g. in regression tests.
- Configurable status messages for use in continous integration environments, e.g. TeamCity.
- Reduces amount of code in tests, which makes them faster to write and easier to read.

It is intended that PlTbUtils will constantly expand by adding more and more functions, procedures and testbench components. Comments, feedback and suggestions are welcome to pela@opencores.org .

The project page on the web is http://opencores.org/project,pltbutils .

Acknowledgements

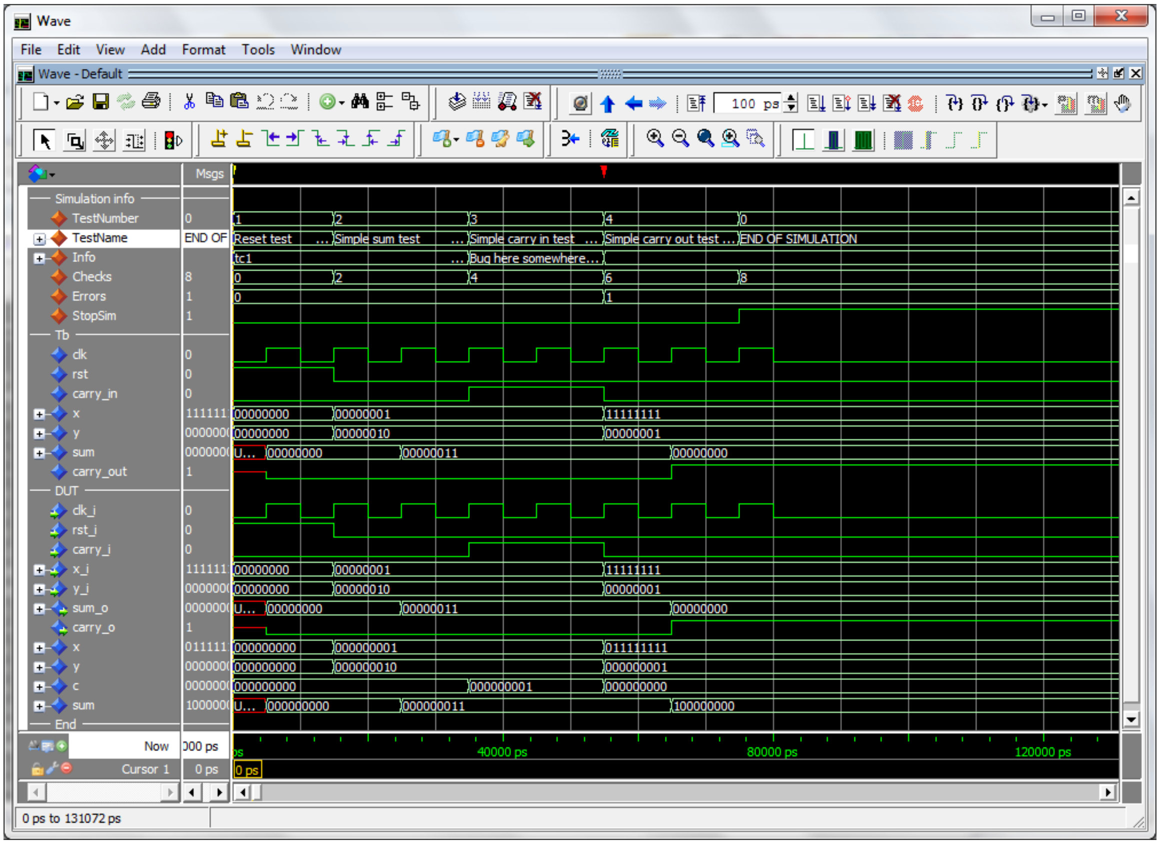
PlTbUtils contains the file txt_util.vhd by Stefan Doll and James F. Frenzel.

Language

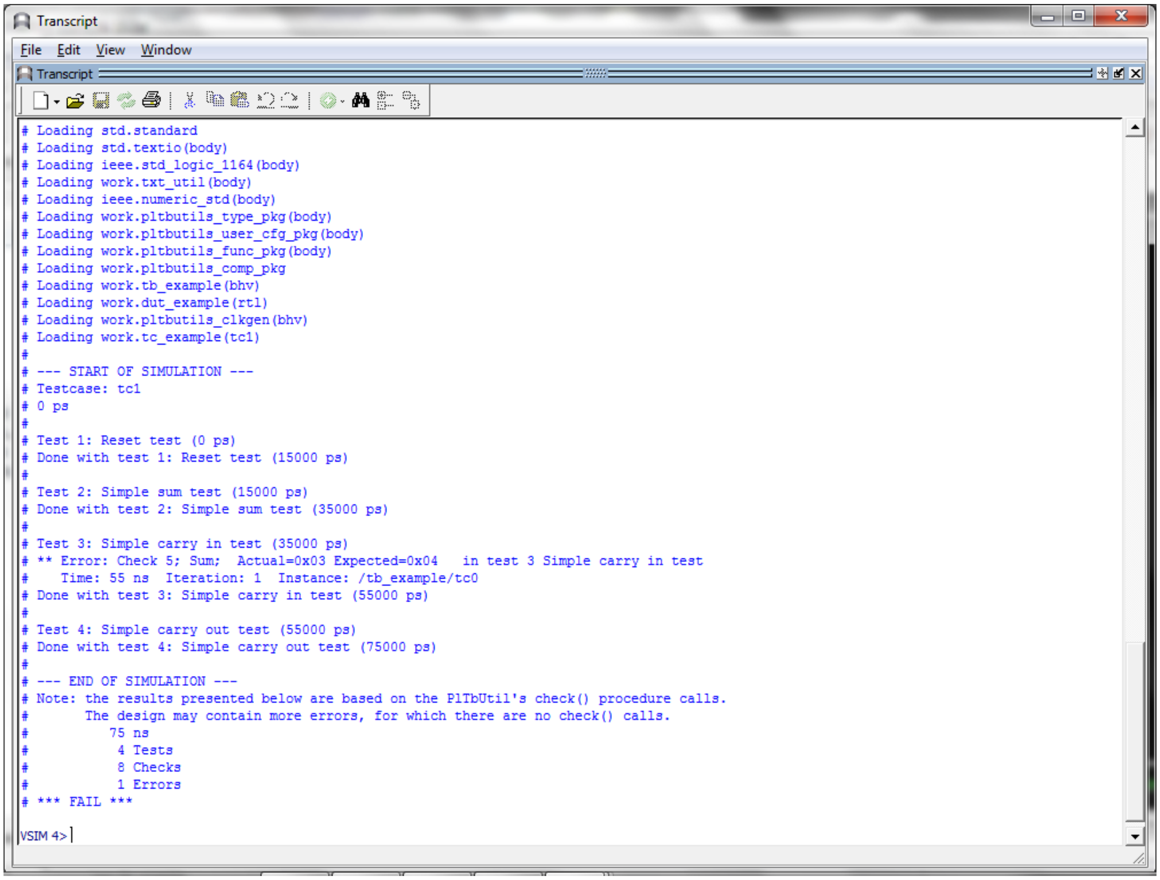
PlTbUtils complies with VHDL-1993, so it works with most VHDL simulators.

However, it is possible to configure the way a simulation stops, by taking advantage of the VHDL-2008 keywords `stop` and `finish`. If your simulator supports `stop` and/or `finish`, see Configuring Simulation Halt on page 18.

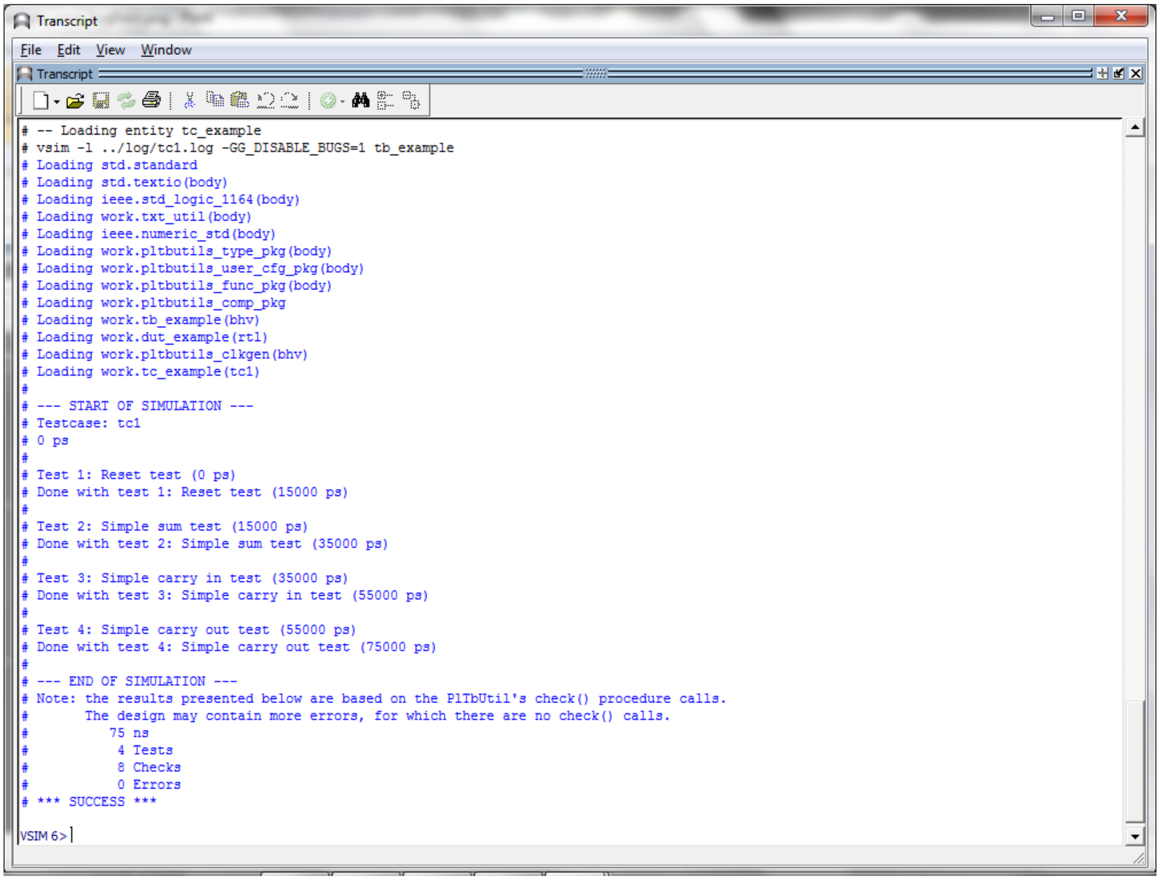
A quick look



During a simulation, the waveform window shows current test number, test name, user-defined info, accumulated number och checks and errors. When the error counter increments, a bug has been found in that point in time.



The transcript window clearly shows points in time where the simulation starts, ends, and where errors are detected. The simulation stops with a clear SUCCESS/FAIL message, specifically formatted for parsing by scripts.




```
# -- Loading entity tc_example
# vsim -l ../log/tc1.log -GG_DISABLE_BUGS=1 tb_example
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.txt_util(body)
# Loading ieee.numeric_std(body)
# Loading work.pltbutils_type_pkg(body)
# Loading work.pltbutils_user_cfg_pkg(body)
# Loading work.pltbutils_func_pkg(body)
# Loading work.pltbutils_comp_pkg
# Loading work.tb_example(bhv)
# Loading work.dut_example(rtl)
# Loading work.pltbutils_clkgen(bhv)
# Loading work.tc_example(tc1)
#
# --- START OF SIMULATION ---
# Testcase: tc1
# 0 ps
#
# Test 1: Reset test (0 ps)
# Done with test 1: Reset test (15000 ps)
#
# Test 2: Simple sum test (15000 ps)
# Done with test 2: Simple sum test (35000 ps)
#
# Test 3: Simple carry in test (35000 ps)
# Done with test 3: Simple carry in test (55000 ps)
#
# Test 4: Simple carry out test (55000 ps)
# Done with test 4: Simple carry out test (75000 ps)
#
# --- END OF SIMULATION ---
# Note: the results presented below are based on the PlTbUtil's check() procedure calls.
#       The design may contain more errors, for which there are no check() calls.
#          75 ns
#           4 Tests
#           8 Checks
#           0 Errors
# *** SUCCESS ***
```

The testcase code is compact and to the point, which results in less code to write, and makes the code easier to read, as in the following example.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.pltbutils_func_pkg.all;

-- NOTE: The purpose of the following code is to demonstrate some of the
-- features in PlTbUtils, not to do a thorough verification.
architecture tc1 of tc_example is
begin
  p_tc1 : process
    variable pltbv  : pltbv_t := C_PLTBV_INIT;
  begin
    startsim("tc1", pltbv, pltbs);
    rst         <= '1';
    carry_in    <= '0';
    x           <= (others => '0');
    y           <= (others => '0');

    starttest(1, "Reset test", pltbv, pltbs);
    waitclks(2, clk, pltbv, pltbs);
    check("Sum during reset",       sum,         0, pltbv, pltbs);
    check("Carry out during reset", carry_out, '0', pltbv, pltbs);
    rst         <= '0';
    endtest(pltbv, pltbs);

    starttest(2, "Simple sum test", pltbv, pltbs);
    carry_in <= '0';
    x <= std_logic_vector(to_unsigned(1, x'length));
    y <= std_logic_vector(to_unsigned(2, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         3, pltbv, pltbs);
    check("Carry out", carry_out, '0', pltbv, pltbs);
    endtest(pltbv, pltbs);

    starttest(3, "Simple carry in test", pltbv, pltbs);
    print(G_DISABLE_BUGS=0, pltbv, pltbs, "Bug here somewhere");
    carry_in <= '1';
    x <= std_logic_vector(to_unsigned(1, x'length));
    y <= std_logic_vector(to_unsigned(2, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         4, pltbv, pltbs);
    check("Carry out", carry_out, '0', pltbv, pltbs);
    print(G_DISABLE_BUGS=0, pltbv, pltbs, "");
    endtest(pltbv, pltbs);

    starttest(4, "Simple carry out test", pltbv, pltbs);
    carry_in <= '0';
    x <= std_logic_vector(to_unsigned(2**G_WIDTH-1, x'length));
    y <= std_logic_vector(to_unsigned(1, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         0, pltbv, pltbs);
    check("Carry out", carry_out, '1', pltbv, pltbs);
    endtest(pltbv, pltbs);

    endsim(pltbv, pltbs, true);
    wait;
  end process p_tc1;
end architecture tc1;
```

2

Tutorial

Basics

We will demonstrate how to use PlTbUtils by showing an example. In this example, we have a DUT (Device Under Test / Design Under Test) with the following entity.

```
entity dut_example is
  generic (
    G_WIDTH          : integer := 8;
    G_DISABLE_BUGS   : integer range 0 to 1 := 1
  );
  port (
    clk_i            : in  std_logic;
    rst_i            : in  std_logic;
    carry_i          : in  std_logic;
    x_i              : in  std_logic_vector(G_WIDTH-1 downto 0);
    y_i              : in  std_logic_vector(G_WIDTH-1 downto 0);
    sum_o            : out std_logic_vector(G_WIDTH-1 downto 0);
    carry_o          : out std_logic
  );
end entity dut_example;
```

As you can see, it has a clock- and a reset input port (clk_i and rst_i), three other input ports (x_i, y_i, and carry_i), and two output ports (sum_o and carry_o). There is also a generic, G_WIDTH, which sets the number of bits in x_i, y_i and sum_o. The second generic, G_DISABLE_BUGS, is very unusual in real designs, but it is useful in this example. We will reveal the purpose of this strange generic later, although some may already be able to guess what it is for.

To verify this DUT, we want the testbench to apply different stimuli to the input ports, and check the response of the output ports. The following code is an example of such a testbench. We will first show all of the code, and then explain parts of it.

```
library ieee;
use ieee.std_logic_1164.all;
use work.pltbutils_func_pkg.all;
use work.pltbutils_comp_pkg.all;

entity tb_example is
  generic (
    G_WIDTH             : integer := 8;
    G_CLK_PERIOD        : time := 10 ns;
    G_DISABLE_BUGS      : integer range 0 to 1 := 0
  );
end entity tb_example;

architecture bhv of tb_example is

  -- Simulation status- and control signals
  -- for accessing .stop_sim and for viewing in waveform window
  signal pltbs          : pltbs_t := C_PLTBS_INIT;

  -- DUT stimuli and response signals
  signal clk            : std_logic;
  signal rst            : std_logic;
  signal carry_in       : std_logic;
  signal x              : std_logic_vector(G_WIDTH-1 downto 0);
  signal y              : std_logic_vector(G_WIDTH-1 downto 0);
  signal sum            : std_logic_vector(G_WIDTH-1 downto 0);
  signal carry_out      : std_logic;

begin

  dut0 : entity work.dut_example
    generic map (
      G_WIDTH           => G_WIDTH,
      G_DISABLE_BUGS    => G_DISABLE_BUGS
    )
    port map (
      clk_i             => clk,
      rst_i             => rst,
      carry_i           => carry_in,
      x_i               => x,
      y_i               => y,
      sum_o             => sum,
      carry_o           => carry_out
    );

  clkgen0 : pltbutils_clkgen
    generic map(
      G_PERIOD          => G_CLK_PERIOD
    )
    port map(
      clk_o             => clk,
      stop_sim_i        => pltbs.stop_sim
    );

  tc0 : entity work.tc_example
    generic map (
      G_WIDTH           => G_WIDTH,
      G_DISABLE_BUGS    => G_DISABLE_BUGS
    )
    port map(
      pltbs             => pltbs,
      clk               => clk,
      rst               => rst,
      carry_in          => carry_in,
      x                 => x,
      y                 => y,
      sum               => sum,
      carry_out         => carry_out
    );
end architecture bhv;
```

As the testbench example shows, the following packages are needed (in addition to the usual std_logic_1164, etc):

```
use work.pltbutils_func_pkg.all;
use work.pltbutils_comp_pkg.all;
```

pltbutils_func_pkg contains type definitions, functions and procedures for controlling stimuli and checking response.

pltbutils_comp_pkg contains component declarations for testbench components.

PlTbUtils uses a variable called pltbv, and a signal called pltbs, for controlling the simulation and keeping track of status. The pltbs signal is useful for viewing in the simulator's waveform window. pltbs is a record containing a number of members which show various information. Expand pltbs in the simulator's waveform window to expose the members. To make it prettier, you can make use of ModelSim's Combine Signals feature. Each member of the pltbs record can be set to be its own Combined Signal, see the waveform images in this document. Other simulators usually have similar features.

The DUT is instansiated in the testbench, as well as a clock generator component from PlTbUtils. We also instansiate a testcase component (tc_example). This testcase component has an entity defined in one file, and the architecture defined in another file. This makes it possible to have several different testcases for the same testbench. Just compile the testcase architecture that you want to use for a specific simulation run.

The entity declaration for the testcase looks as follows.

```
library ieee;
use ieee.std_logic_1164.all;
use work.pltbutils_func_pkg.all;

entity tc_example is
  generic (
    G_WIDTH         : integer := 8;
    G_DISABLE_BUGS  : integer range 0 to 1 := 0
  );
  port (
    pltbs           : out pltbs_t;
    clk             : in  std_logic;
    rst             : out std_logic;
    carry_in        : out std_logic;
    x               : out std_logic_vector(G_WIDTH-1 downto 0);
    y               : out std_logic_vector(G_WIDTH-1 downto 0);
    sum             : in  std_logic_vector(G_WIDTH-1 downto 0);
    carry_out       : in  std_logic
  );
end entity tc_example;
```

The ports of the testcase components are the same as for the DUT, but the mode (direction) of the ports are the opposite, so the testcase component can drive the inputs of the DUT, and detect the values of the output of the DUT. The only exception to this rule is the clock, which is an input, just as for the DUT.

There is also an output port for pltbs, because pltbs is driven from the tc architecture.

One possible testcase architecture could look as the following code.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.txt_util.all;
use work.pltbutils_func_pkg.all;

-- NOTE: The purpose of the following code is to demonstrate some of the
-- features in PlTbUtils, not to do a thorough verification.
architecture tc1 of tc_example is
begin
  p_tc1 : process
    variable pltbv  : pltbv_t := C_PLTBV_INIT;
  begin
    startsim("tc1", pltbv, pltbs);
    rst         <= '1';
    carry_in    <= '0';
    x           <= (others => '0');
    y           <= (others => '0');

    starttest(1, "Reset test", pltbv, pltbs);
    waitclks(2, clk, pltbv, pltbs);
    check("Sum during reset",       sum,         0, pltbv, pltbs);
    check("Carry out during reset", carry_out, '0', pltbv, pltbs);
    rst         <= '0';
    endtest(pltbv, pltbs);

    starttest(2, "Simple sum test", pltbv, pltbs);
    carry_in <= '0';
    x <= std_logic_vector(to_unsigned(1, x'length));
    y <= std_logic_vector(to_unsigned(2, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         3, pltbv, pltbs);
    check("Carry out", carry_out, '0', pltbv, pltbs);
    endtest(pltbv, pltbs);

    starttest(3, "Simple carry in test", pltbv, pltbs);
    print(G_DISABLE_BUGS=0, pltbv, pltbs, "Bug here somewhere");
    carry_in <= '1';
    x <= std_logic_vector(to_unsigned(1, x'length));
    y <= std_logic_vector(to_unsigned(2, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         4, pltbv, pltbs);
    check("Carry out", carry_out, '0', pltbv, pltbs);
    print(G_DISABLE_BUGS=0, pltbv, pltbs, "");
    endtest(pltbv, pltbs);

    starttest(4, "Simple carry out test", pltbv, pltbs);
    carry_in <= '0';
    x <= std_logic_vector(to_unsigned(2**G_WIDTH-1, x'length));
    y <= std_logic_vector(to_unsigned(1, x'length));
    waitclks(2, clk, pltbv, pltbs);
    check("Sum",       sum,         0, pltbv, pltbs);
    check("Carry out", carry_out, '1', pltbv, pltbs);
    endtest(pltbv, pltbs);

    endsim(pltbv, pltbs, true);
    wait;
  end process p_tc1;
end architecture tc1;
```

The testcase process starts with calling the procedure startsim(). This procedure clears pltbv and pltbs, and outputs a message to the transcript and to the waveform window to inform that the simulation now starts. The first argument to startsim is the name of the testcase.

The last arguments of startsim(), and to many other procedures in PlTbUtils, are pltbv and pltbs.

After initiating stimuli to the DUT, we call the procedure starttest() with the number and name for the first test. starttest() prints the test number and test name to the transcript and to the waveform window, and updates pltbv and pltbs.

Then we need to wait until the DUT has reacted to the stimuli. We do this by calling the procedure waitclks(), which waits a specified number of cycles of the specified clock.

After this, we start checking the results, by examining the outputs from the DUT. To do this, we use the check() procedure. The first argument is a text string that specifies what we check, the second argument is the signal or variable that we want to examine, and the third is the expected value of the signal or variable. If the examined signal holds the expected value, nothing is printed. But if the value is incorrect, the string in the first argument is printed, together with the actual and expected values of the signal. The number and name of the test (as specified with starttest()) is also printed. PlTbUtils' check counter is incremented for every check() procedure call, and the error counter is incremented in case of error.

After the test, we call endtest().

We make a number of different tests by calling starttest(), setting stimuli, waiting for the DUT to react with waitclks() or some other means, and checking the outputs with the check() procedure, and calling endtest().

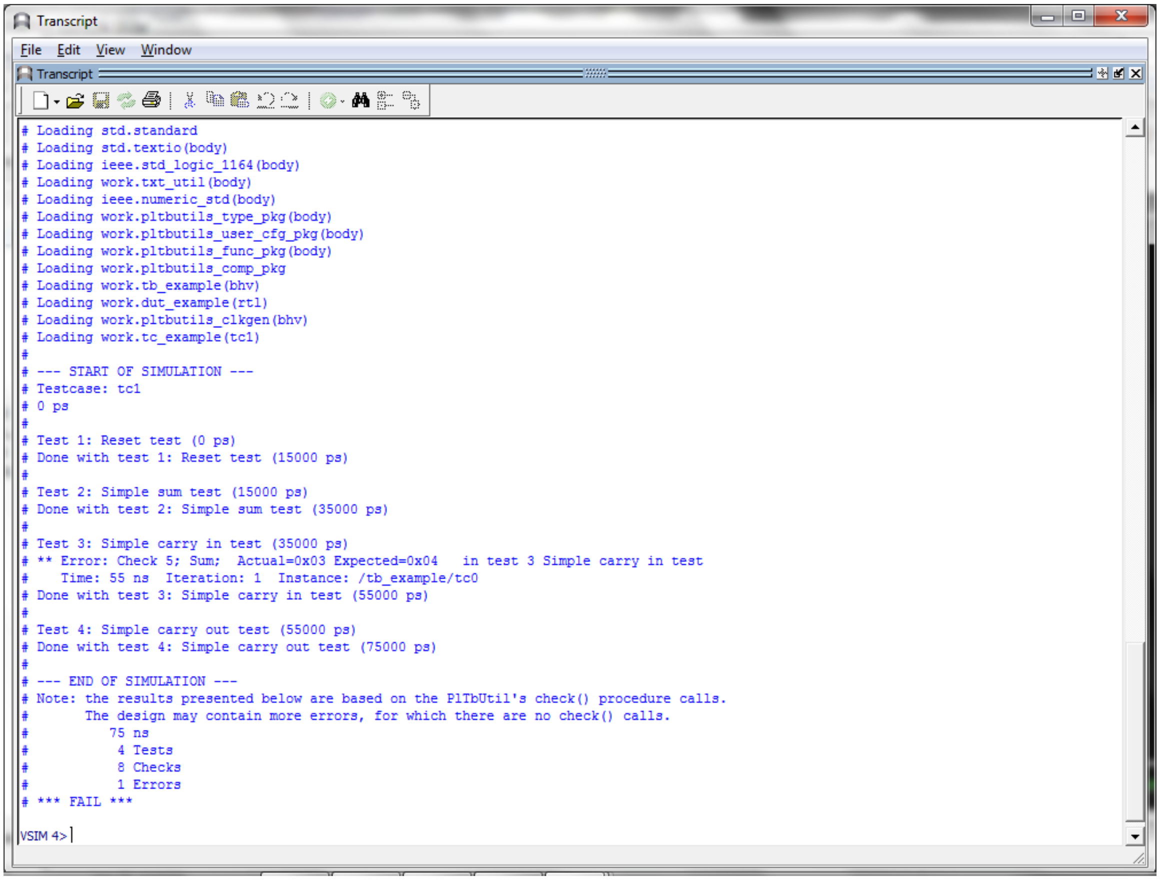
Finally, we call the endsim() procedure, which prints an end-of-simulation message to the transcript, and presents the results, including a SUCCESS or FAIL message.

The start-of-simulation message, end-of-simulation message, and SUCCESS/FAIL messages are unique, to make them easy to search for by scripts. This simplifies collection of simulation status for regression tests with a lot of different simulations.

Try it out in your simulator! The pltbutils files that need to be compiled are located in src/vhdl/, and they are listed in compile order in pltbutils_files.lst . The example DUT and example testbench files are located in example/vhdl/, and the files are listed in compile order in example_dut.lst and tb_example_files.lst .

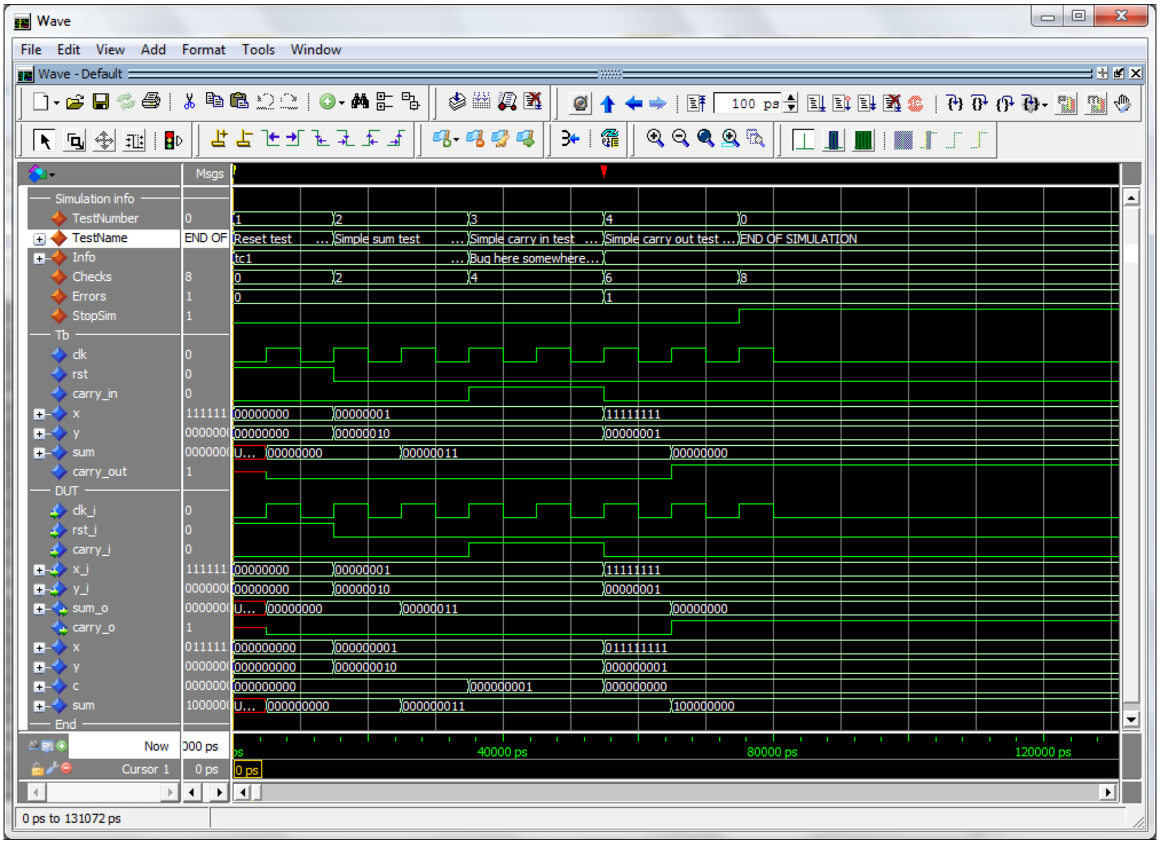
If you are a ModelSim user, there are .do files available in sim/example_sim/run/ . To use them, start Start ModelSim, and in the ModelSim Gui select the menu item File->Change directory... . Navigate to the PlTbUtils directory sim/example_sim/run/ and click Ok. Then, in the transcript window, type do run_tc1.do .

The simulation will start, and the transcript from the simulation looks as follows.



The transcript says that one error has been found at 55 ns, in test 3; Simple carry in test.

The waveform window looks like this.



Here we can see the error detected at the point in time where the error counter increments from 0 to 1. Again, we can that the error is found in test 3, the Simple carry in test.

Have a look at the DUT code in example/vhdl/dut_example.vhd . It looks as follows.

```
x <= resize(unsigned(x_i), G_WIDTH+1);
y <= resize(unsigned(y_i), G_WIDTH+1);
c <= resize(unsigned(std_logic_vector'('0' & carry_i)), G_WIDTH+1);

p_sum : process(clk_i)
begin
  if rising_edge(clk_i) then
    if rst_i = '1' then
      sum <= (others => '0');
    else
      if G_DISABLE_BUGS = 1 then
        sum <= x + y + c;
      else
        sum <= x + y;
      end if;
    end if;
  end if;
end process;
```

The code really looks suspisious. If the generic G_DISABLE_BUGS is not one, the carry input is not added to the sum. But we need the carry input to be added to the sum!

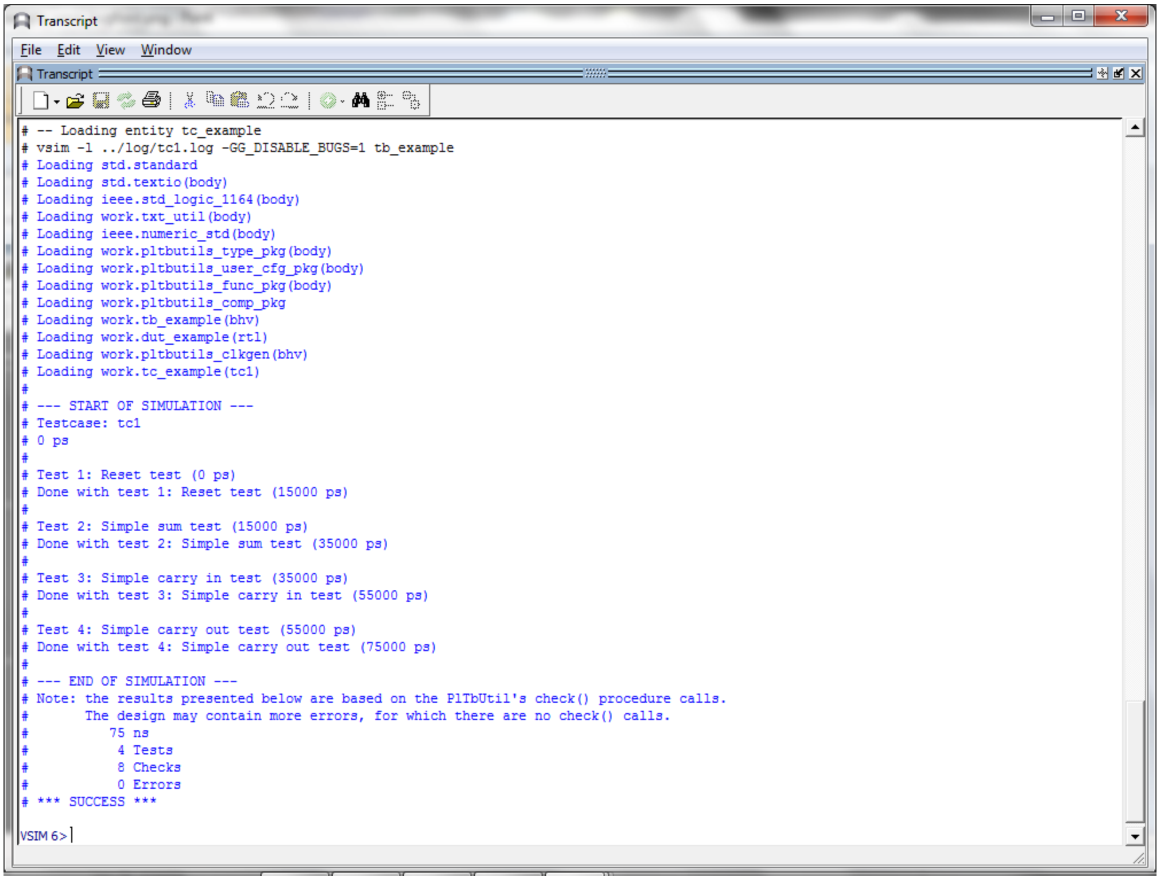
A simple way do disable this bug, is to set the generic G_DISABLE_BUGS to one. In this case, this can be done very easily, without any modifying and code.

In the ModelSim transcript window, type

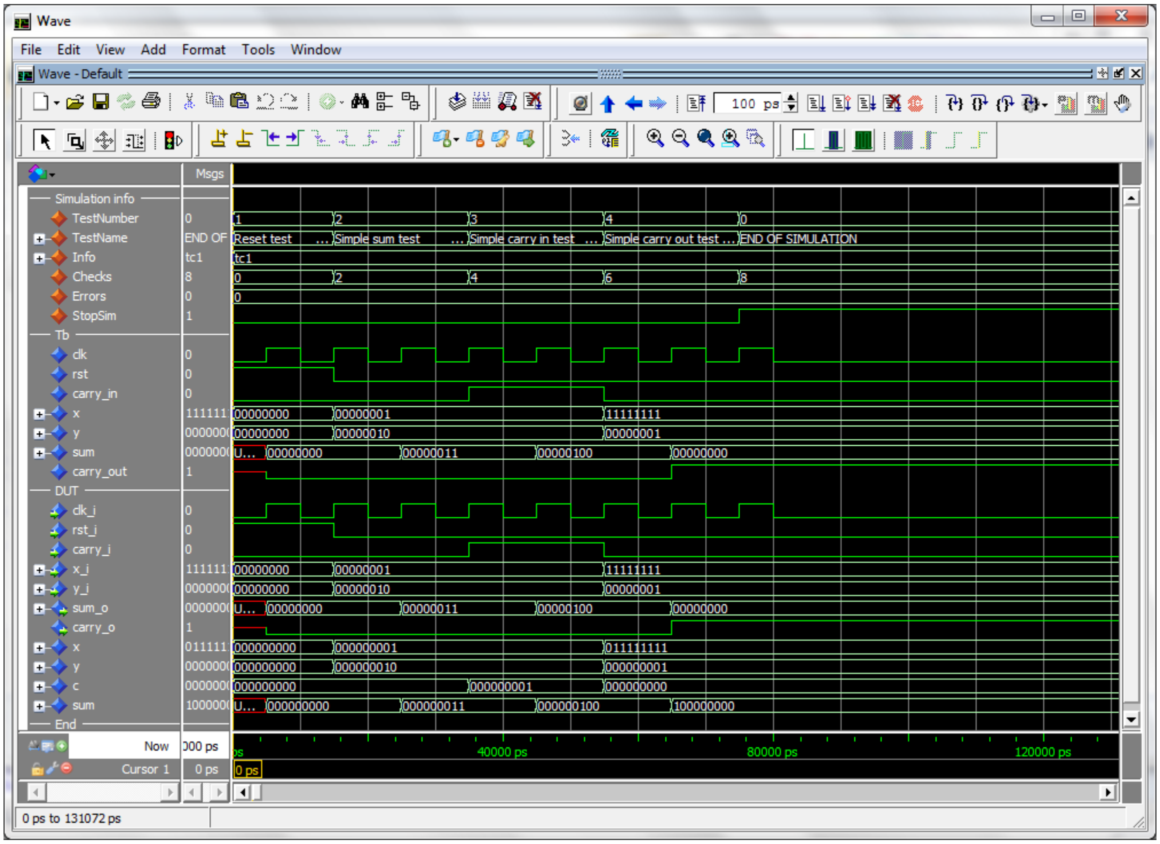
do run_tc1_bugfixed.do

This will run the test again, but now with the generic G_DISABLE_BUGS set to 1.

The transcript and waveform windows will now look like the following images.



Transcript
File Edit View Window
Transcript
-- Loading entity tc_example
vsim -l ../log/tc1.log -GG_DISABLE_BUGS=1 tb_example
Loading std.standard
Loading std.textio(body)
Loading ieee.std_logic_1164(body)
Loading work.txt_util(body)
Loading ieee.numeric_std(body)
Loading work.pltbutils_type_pkg(body)
Loading work.pltbutils_user_cfg_pkg(body)
Loading work.pltbutils_func_pkg(body)
Loading work.pltbutils_comp_pkg
Loading work.tb_example(bhv)
Loading work.dut_example(rtl)
Loading work.pltbutils_clkgen(bhv)
Loading work.tc_example(tc1)
#
--- START OF SIMULATION ---
Testcase: tc1
0 ps
#
Test 1: Reset test (0 ps)
Done with test 1: Reset test (15000 ps)
#
Test 2: Simple sum test (15000 ps)
Done with test 2: Simple sum test (35000 ps)
#
Test 3: Simple carry in test (35000 ps)
Done with test 3: Simple carry in test (55000 ps)
#
Test 4: Simple carry out test (55000 ps)
Done with test 4: Simple carry out test (75000 ps)
#
--- END OF SIMULATION ---
Note: the results presented below are based on the PlTbUtil's check() procedure calls.
The design may contain more errors, for which there are no check() calls.
75 ns
4 Tests
8 Checks
0 Errors
*** SUCCESS ***
VSIM 6>



This tutorial has shown some of the available procedures and testbench components in PlTbUtils. For a complete list, see the reference section.

When you want to make your own testbenches with PlTbUtils, have a look at the template files in template/vhdl/ .

User Configuration

It is possible to configure some aspects of PlTbUtils's behaviour, by modifying the package file pltbutils_user_cfg.pkg.

It is recommended NOT to modify the file directly. Instead, copy it to another directory and modify the copy. Make the simulator read the modified copy instead of the original. This makes it easier to update pltbutils to a later version without destroying the modifications. After updating, check if antyhing has changed in the file, and change your modified copy accordingly.

Configuring Simulation Halt

When calling endsim(), the signal stop_sim is set to '1'. When set, all clock generators etc in the testbench and the DUT should stop, so there will be no further events in the simulation. The simulator will detect that nothing more will happen, and stops the simulation.

In some cases, it is not possible to stop clock generators, PLL models etc. In that case, endsim() can force a simulaton halt, by setting the force argument to true.

The declaration of endsim() is

```
procedure endsim(
  signal   pltbutils_sc       : out pltbutils_sc_t;
  constant show_success_fail  : in  boolean := false;
  constant force              : in  boolean := false
);
```

so to force a simulation halt, call endsim with

```
endsim(pltbutils_sc, true, true);
```

This stops the simulationg using an assert-failure. This works in all versions of VHDL, but it is an ugly way of doing it, since it outputs a failure message for something which isn't a failure.

You can change the way the simulation stops when the force flag is set in your copy of pltbutils_user_cfg.vhd.

Change the constant `C_PLTBUTILS_USE_CUSTOM_STOPSIM` to true, and modify the behaviour of the procedure `custom_stopsim()`. In VHDL-2008 the new keywords stop and finish was introduced. Try one of them, for example.

Configuring Messages for Integration Environments

It is possible adapt the status messages to suit various continous integration environments, e.g. TeamCity, by specifying what the messages should look like.

You can create your own messages printed when starting and stopping a simulation, starting and stopping a test, for checking, etc.

In your copy of pltbutils_user_cfg_pkg.vhd, set one or more of the message constants to true, and modify the associated procedure.

The constants are

```
C_PLTBUTILS_USE_CUSTOM_STARTSIM_MSG
C_PLTBUTILS_USE_CUSTOM_ENDSIM_MSG
C_PLTBUTILS_USE_CUSTOM_STARTTEST_MSG
C_PLTBUTILS_USE_CUSTOM_ENDTEST_MSG
C_PLTBUTILS_USE_CUSTOM_CHECK_MSG
C_PLTBUTILS_USE_CUSTOM_ERROR_MSG
```

The corresponding procedures already contain examples for TeamCity. Modify if you use another environment.

You can disable the standard messages by setting the standard constants to false (`C_PLTBUTILS_USE_STD_STARTSIM_MSG` etc).

3

Reference

PlTbUtils files

The PlTbUtils files are located in src/vhdl/ .
The files needed to be compiled are listed in compile order in pltbutils_files.lst .

See example testbench using PlTbUtils in example/vhdl/ .
This code can be simulated from sim/example_sim/run/ .

Template code is available in template/vhdl/ .

Functions and procedures

startsim

```
procedure startsim(
   constant testcase_name      : in    string;
   variable pltbv              : inout pltbv_t;
   signal   pltbs              : out   pltbs_t
)
```

Displays a message at start of simulation message, and initializes PlTbUtils' status and control variable and -signal. Call startsim() only once.

Arguments:

testcase_name	Name of the test case, e.g. "tc1".
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.

The start-of-simulation message is not only intended to be informative for humans. It is also intended to be searched for by scripts, e.g. for collecting results from a large number of regression tests.

Example:

```
startsim("tc1", pltbutils_sc);
```

endsim

```
procedure endsim(
  variable pltbv               : inout pltbv_t;
  signal   pltbs               : out   pltbs_t;
  constant show_success_fail   : in  boolean := false;
  constant force               : in  boolean := false
)
```

Displays a message at end of simulation message, presents the simulation results, and stops the simulation. Call endsim() it only once.

Arguments:

pltbv, pltbs	PlTbUtils' status- and control variable and -signal.
show_success_fail	If true, endsim() shows "*** SUCCESS ***", "*** FAIL ***", or "*** NO CHECKS ***". Optional, default is false.
force	If true, forces the simulation to stop using an assert failure statement. Use this option only if the normal way of stopping the simulation doesn't work (see below). Optional, default is false.

The testbench should be designed so that all clocks stop when endsim() sets the signal stop_sim to '1'. This should stop the simulator.

In some cases, that doesn't work, then set the force argument to true, which causes a false assert failure, which should stop the simulator.

The end-of-simulation messages and success/fail messages are not only intended to be informative for humans. They are also intended to be searched for by scripts, e.g. for collecting results from a large number of regression tests.

Examples:

```
endsim(pltbutils_sc);
endsim(pltbutils_sc, true);
endsim(pltbutils_sc, true, true);
```

starttest

```
procedure starttest(
  constant num                : in    integer := -1;
  constant name               : in    string;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t
)
```

Sets a number (optional) and a name for a test. The number and name will be printed to the screen, and displayed in the simulator's waveform window.

The test number and name is also included if there errors reported by the check() procedure calls.

Arguments:

num	Test number. Optional, default is to increment the current test number.
name	Test name.
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.

If the test number is omitted, a new test number is automatically computed by incrementing the current test number. Manually setting the test number may make it easier to find the test code in the testbench code, though.

Examples:

```
starttest("Reset test", pltbv, pltbs);
starttest(1, "Reset test", pltbv, pltbs);
```

endtest

```
procedure endtest(
  variable pltbv                : inout pltbv_t;
  signal   pltbs                : out   pltbs_t
)
```

Prints an end-of-test message to the screen.

Arguments:

pltbv, pltbs	PlTbUtils' status- and control variable and -signal.

Example:

```
endtest(pltbv, pltbs);
```

print printv print2

```
procedure print(
  signal   s                  : out   string;
  constant txt                : in    string
)

procedure print(
  constant active             : in    boolean;
  signal   s                  : out   string;
  constant txt                : in    string
)

procedure print(
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t;
  constant txt                : in    string
)

procedure print(
  constant active             : in    boolean;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t;
  constant txt                : in    string
)

procedure printv(
  variable s                  : out   string;
  constant txt                : in    string
)

procedure printv(
  constant active             : in    boolean;
  variable s                  : out   string;
  constant txt                : in    string
)
```

```
procedure print2(
  signal   s                  : out   string;
  constant txt                : in    string
)

procedure print2(
  constant active             : in    boolean;
  signal   s                  : out   string;
  constant txt                : in    string
)

procedure print2(
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t;
  constant txt                : in    string
)

procedure print2(
  constant active             : in    boolean;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t
  constant txt                : in    string
)
```

print() prints text messages to a signal for viewing in the simulator's waveform window. printv() does the same thing, but to a variable instead. print2() prints both to a signal and to the transcript window.

The type of the output can be string or pltbv+pltbs.

Arguments:

s	Signal or variable of type string to be printed to.
txt	The text.
active	The text is only printed if active is true. Useful for debug switches, etc.
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.

If the string txt is longer than the signal s, the text will be truncated. If txt is shorter, s will be padded with spaces.

NOTE: more print procedures are available in txt_util.txt .

Examples:

```
print(msg, "Hello, world"); -- Prints to signal msg

print(G_DEBUG, msg, "Hello, world"); -- Prints to signal msg if
                                     -- generic G_DEBUG is true

printv(v_msg, "Hello, world"); -- Prints to variable msg

print(pltbv, pltbs, "Hello, world"); -- Prints to "info" in waveform
                                     -- window

print2(msg, "Hello, world"); -- Prints to signal and transcript window

print(pltbv, pltbs, "Hello, world"); -- Prints to "info" in waveform and

                                     -- transcript windows
```

waitclks

```
procedure waitclks(
  constant n                    : in    natural;
  signal   clk                  : in    std_logic;
  variable pltbv                : inout pltbv_t;
  signal   pltbs                : out   pltbs_t;
  constant falling              : in    boolean := false;
  constant timeout              : in    time    := C_PLTBUTILS_TIMEOUT
)
```

Waits specified amount of clock cycles of the specified clock. Or, to be more precise, a specified number of specified clock edges of the specified clock.

Arguments:

n	Number of rising or falling clock edges to wait.
clk	The clock to wait for.
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.
falling	If true, waits for falling edges, otherwise rising edges. Optional, default is false.
timeout	Timeout time, in case the clock is not working. Optional, default is C_PLTBUTILS_TIMEOUT.

Examples:

```
waitclks(5, sys_clk, pltbv, pltbs);
waitclks(5, sys_clk, pltbv, pltbs, true);
waitclks(5, sys_clk, pltbv, pltbs, true, 1 ms);
```

waitsig

```
procedure waitsig(
  signal   s                   : in
           integer|std_logic|std_logic_vector|unsigned|signed;
  constant value               : in
           integer|std_logic|std_logic_vector|unsigned|signed;
  signal   clk                 : in    std_logic;
  variable pltbv               : inout pltbv_t;
  signal   pltbs               : out   pltbs_t;
  constant falling             : in    boolean := false;
  constant timeout             : in    time    := C_PLTBUTILS_TIMEOUT)
```

Waits until a signal has reached a specified value after specified clock edge.

Arguments:

s	The signal to test. Supported types: integer, std_logic, std_logic_vector, unsigned, signed.
value	Value to wait for. Same type as data or integer.
clk	The clock.
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.
falling	If true, waits for falling edges, otherwise rising edges. Optional, default is false.
timeout	Timeout time, in case the clock is not working. Optional, default is C_PLTBUTILS_TIMEOUT.

Examples:

```
waitsig(wr_en, '1', sys_clk, pltbv, pltbs);
waitsig(rd_en,   1, sys_clk, pltbv, pltbs, true);
waitclks(full, '1', sys_clk, pltbv, pltbs, true, 1 ms);
```

check

```
procedure check(
  constant rpt                : in    string;
  constant data               : in    integer |
                                      std_logic | std_logic_vector |
                                      unsigned | signed;
  constant expected           : in    integer |
                                      std_logic | std_logic_vector |
                                      unsigned | signed;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t
)

procedure check(
  constant rpt                : in    string;
  constant data               : in    std_logic_vector;
  constant expected           : in    std_logic_vector;
  constant mask               : in    std_logic_vector;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t
)

procedure check(
  constant rpt                : in    string;
  constant expr               : in    boolean;
  variable pltbv              : inout pltbv_t;
  signal   pltbs              : out   pltbs_t
)
```

Checks that the value of a signal or variable is equal to expected. If not equal, displays an error message and increments the error counter.

Arguments:

rpt	Report message to be displayed in case of mismatch. It is recommended that the message is unique and that it contains the name of the signal or variable being checked. The message should NOT contain the expected value, becase check() prints that automatically.
data	The signal or variable to be checked. Supported types: integer, std_logic, std_logic_vector, unsigned, signed.
expected	Expected value. Same type as data, or integer.
mask	Bit mask and:ed to data and expected before comparison. Optional if data is std_logic_vector. Not allowed for other types.
expr	boolean expression for checking. This makes it possible to check any kind of expresion, not just equality.
pltbv, pltbs	PlTbUtils' status- and control variable and -signal.

Examples:

```
check("dat_o after reset", dat_o, 0, pltbv, pltbs);

-- With mask:
check("Status field in reg_o after start", reg_o, x"01", x"03",
      pltbv, pltbs);

-- Boolean expresson:
check("Counter after data burst", cnt_o > 10, pltbv, pltbs);
```

Testbench components

pltbutils_clkgen

Creates a clock for use in a testbech. The clock stops when input port stop_sim goes ‘1’. This makes the simulator stop (unless there are other infinite processes running in the simulation).

Generic	Width	Type	Description
G_PERIOD	1	time	Clock period.
G_INITVALUE	1	std_logic	Initial value of the non-inverted clock output.

Port	Width	Direction	Description
clk_o	1	Output	Non-inverted clock output. Use this output for single ended or differential clocks.
clk_n_o	1	Output	Inverted clock output. Use if a differential clock is needed, leave open if single-ended clock is needed.
stop_sim_i	1	Input	When ‘1’, stops the clock. This will normally stop the simulation.