Advanced VHDL Verification by combining testing frameworks

VUnit + UVVM + OSVVM

Installing required components

Python - https://www.python.org/downloads/

Git - https://git-scm.com/downloads

Download both installation files and run them as administrator on your system.

We'll use both Python and Git to install subsequent packages

VUnit installation

The easiest way to install is through ***pip***

```
>pip install vunit_hdl
```

Once installed it can be upgraded with the following command:

```
>pip install -U vunit_hdl
```

https://vunit.github.io/installing.html

UVVM installation

Go to https://github.com/OSVVM and download the latest stable version of the package as a zip file.

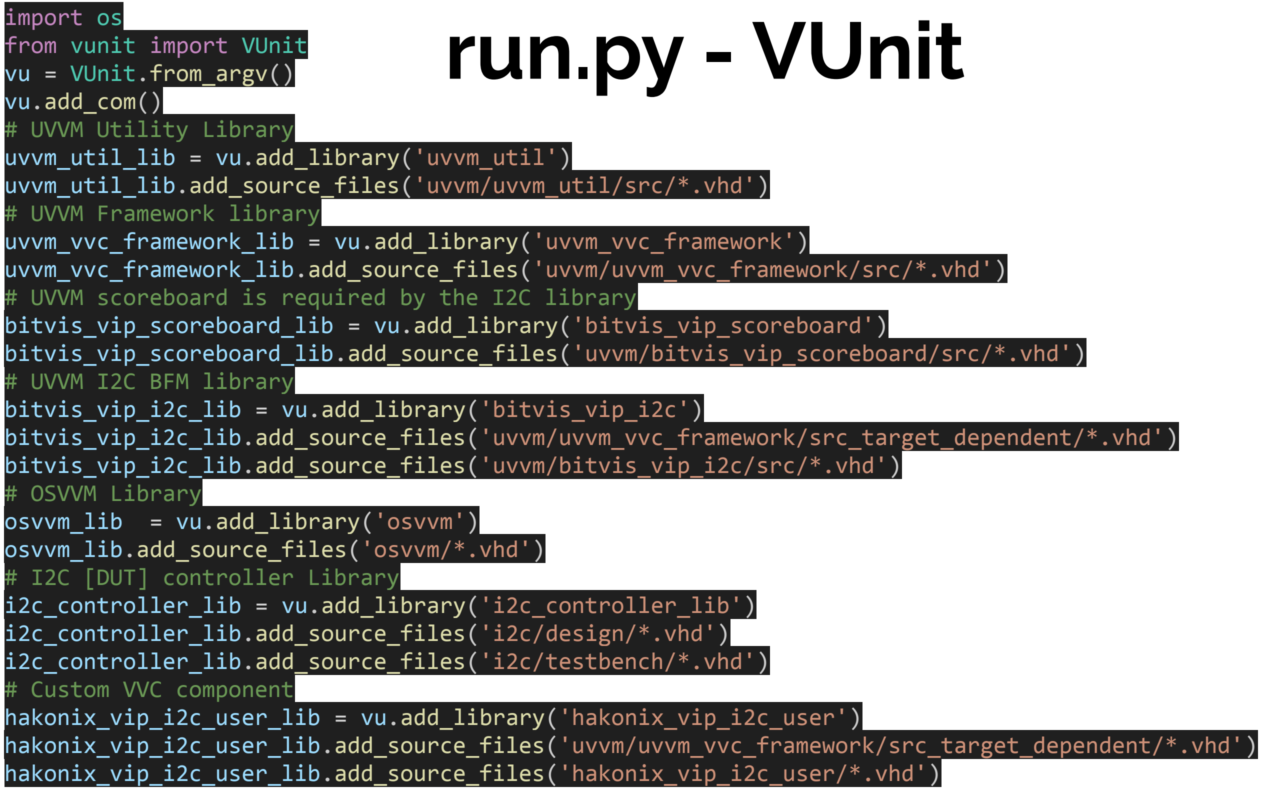
Once downloaded unpack and copy the folder contents onto a folder named **frameworks/uvvm**

OSVVM installation

Go to https://github.com/OSVVM and download the latest stable version of the package as a zip file.

Once downloaded unpack and copy the folder contents onto a folder named **frameworks/osvvm**

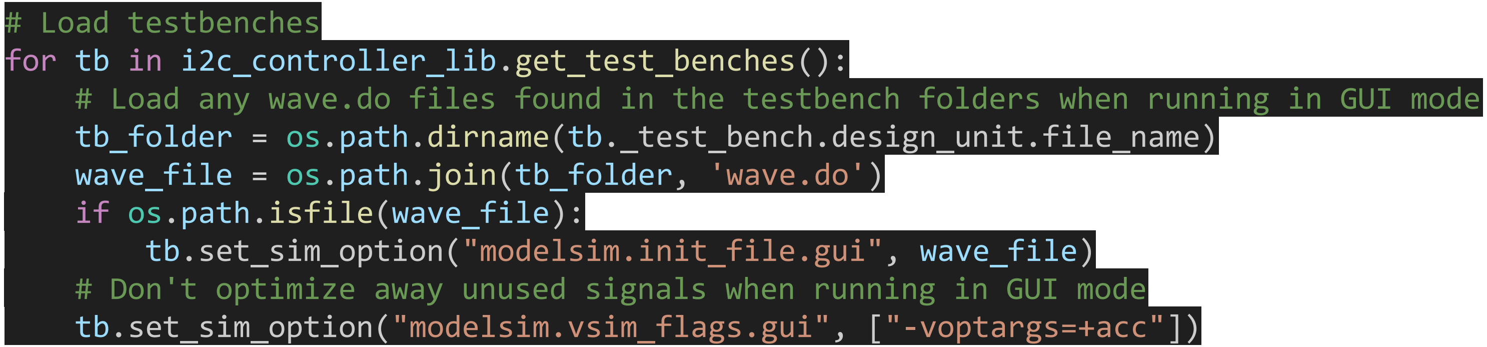
run.py - VUnit



```python
import os
from vunit import VUnit
vu = VUnit.from_argv()
vu.add_com()
# UVVM Utility Library
uvvm_util_lib = vu.add_library('uvvm_util')
uvvm_util_lib.add_source_files('uvvm/uvvm_util/src/*.vhd')
# UVVM Framework library
uvvm_vvc_framework_lib = vu.add_library('uvvm_vvc_framework')
uvvm_vvc_framework_lib.add_source_files('uvvm/uvvm_vvc_framework/src/*.vhd')
# UVVM scoreboard is required by the I2C library
bitvis_vip_scoreboard_lib = vu.add_library('bitvis_vip_scoreboard')
bitvis_vip_scoreboard_lib.add_source_files('uvvm/bitvis_vip_scoreboard/src/*.vhd')
# UVVM I2C BFM library
bitvis_vip_i2c_lib = vu.add_library('bitvis_vip_i2c')
bitvis_vip_i2c_lib.add_source_files('uvvm/uvvm_vvc_framework/src_target_dependent/*.vhd')
bitvis_vip_i2c_lib.add_source_files('uvvm/bitvis_vip_i2c/src/*.vhd')
# OSVVM Library
osvvm_lib  = vu.add_library('osvvm')
osvvm_lib.add_source_files('osvvm/*.vhd')
# I2C [DUT] controller Library
i2c_controller_lib = vu.add_library('i2c_controller_lib')
i2c_controller_lib.add_source_files('i2c/design/*.vhd')
i2c_controller_lib.add_source_files('i2c/testbench/*.vhd')
# Custom VVC component
hakonix_vip_i2c_user_lib = vu.add_library('hakonix_vip_i2c_user')
hakonix_vip_i2c_user_lib.add_source_files('uvvm/uvvm_vvc_framework/src_target_dependent/*.vhd')
hakonix_vip_i2c_user_lib.add_source_files('hakonix_vip_i2c_user/*.vhd')
```

Including Testbenches

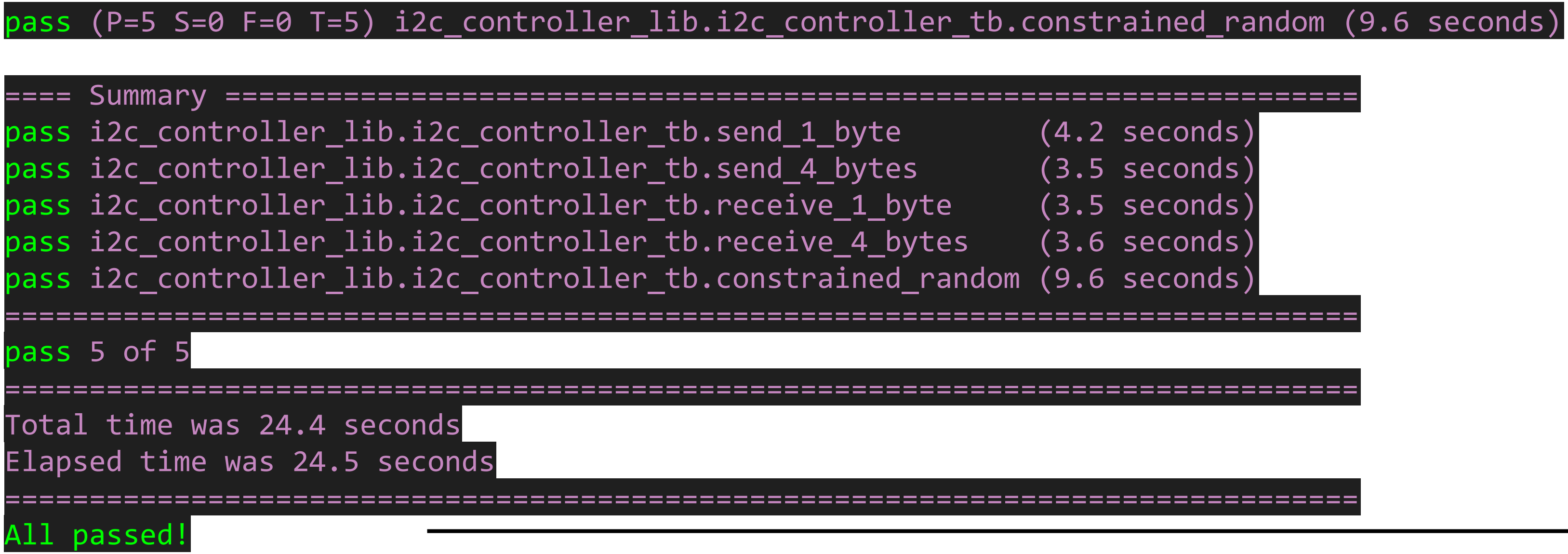
This code will check if test benches include the wave.do file, if so it will be loaded



```
# Load testbenches
for tb in i2c_controller_lib.get_test_benches():
    # Load any wave.do files found in the testbench folders when running in GUI mode
    tb_folder = os.path.dirname(tb._test_bench.design_unit.file_name)
    wave_file = os.path.join(tb_folder, 'wave.do')
    if os.path.isfile(wave_file):
        tb.set_sim_option("modelsim.init_file.gui", wave_file)
    # Don't optimize away unused signals when running in GUI mode
    tb.set_sim_option("modelsim.vsim_flags.gui", ["-voptargs=+acc"])
```

Python script output

The script will compile all required data by typing , and create a folder named **vunit_out**



```
pass (P=5 S=0 F=0 T=5) i2c_controller_lib.i2c_controller_tb.constrained_random (9.6 seconds)

==== Summary ===================================================================
pass i2c_controller_lib.i2c_controller_tb.send_1_byte        (4.2 seconds)
pass i2c_controller_lib.i2c_controller_tb.send_4_bytes       (3.5 seconds)
pass i2c_controller_lib.i2c_controller_tb.receive_1_byte     (3.5 seconds)
pass i2c_controller_lib.i2c_controller_tb.receive_4_bytes    (3.6 seconds)
pass i2c_controller_lib.i2c_controller_tb.constrained_random (9.6 seconds)
================================================================================
pass 5 of 5
================================================================================
Total time was 24.4 seconds
Elapsed time was 24.5 seconds
================================================================================
All passed!
```

VUnit setup on the testbench

Import vunit libraries and context

```
-- VUnit
library vunit_lib;
context vunit_lib.vunit_context;
context vunit_lib.com_context;
```

Add the generic for vunit runner_cfg

```
entity i2c_controller_tb is
  generic(runner_cfg : string); -- VUnit
end i2c_controller_tb;
```

Create the test case(s) and start the runner

Start the test runner object

```
    begin
      --------------------
      -- VUNIT setup
      --------------------
      test_runner_setup(runner, runner_cfg);
```

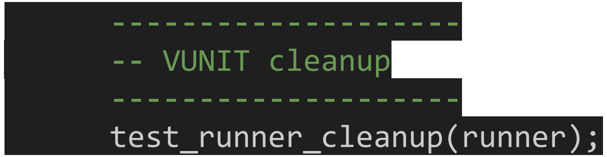
Create test cases by using the **if run(testcase_name) elsif(testcase_name) / end if;**

```
if run("send_1_byte") then
  log(ID_SEQUENCER, "Send 1 byte - i2c_slave_check + i2c_user_transmit");
  i2c_slave_check(I2C_VVCT, 1, x"CD", "Target expecting to receive 1 byte");
  i2c_user_transmit(I2C_USER_VVCT, 1, x"CD", "Controller sending 1 byte");
elsif run("send_2_bytes") then
  log(ID_SEQUENCER, "Send 2 byte - i2c_slave_check+i2c_user_transmit (overloaded) t_byte_array ");
  i2c_slave_check(I2C_VVCT, 1, t_byte_array'(x"12", x"34"), "Target expecting to receive 2 bytes");
  i2c_user_transmit(I2C_USER_VVCT, 1, t_byte_array'(x"12", x"34"), "Controller sending 2 bytes");
...
end if;
```

VUnit cleaner

Test cases on this example are:
send_1_byte // **send_4_bytes** // **receive_1_byte** // **receive_4_bytes** //
constrained_random

Once test cases are completed then VUnit cleanup is required



```
      --------------------
      -- VUNIT cleanup
      --------------------
      test_runner_cleanup(runner);
```

List and run test benches

In order to check which test benches are available type:

```
framewokr_proj> python run.py -l (--list)
i2c_controller_lib.i2c_controller_tb.send_1_byte
i2c_controller_lib.i2c_controller_tb.send_2_bytes
i2c_controller_lib.i2c_controller_tb.receive_1_byte
i2c_controller_lib.i2c_controller_tb.receive_2_bytes
i2c_controller_lib.i2c_controller_tb.constrained_random
Listed 5 tests
```

Run the listed test bench

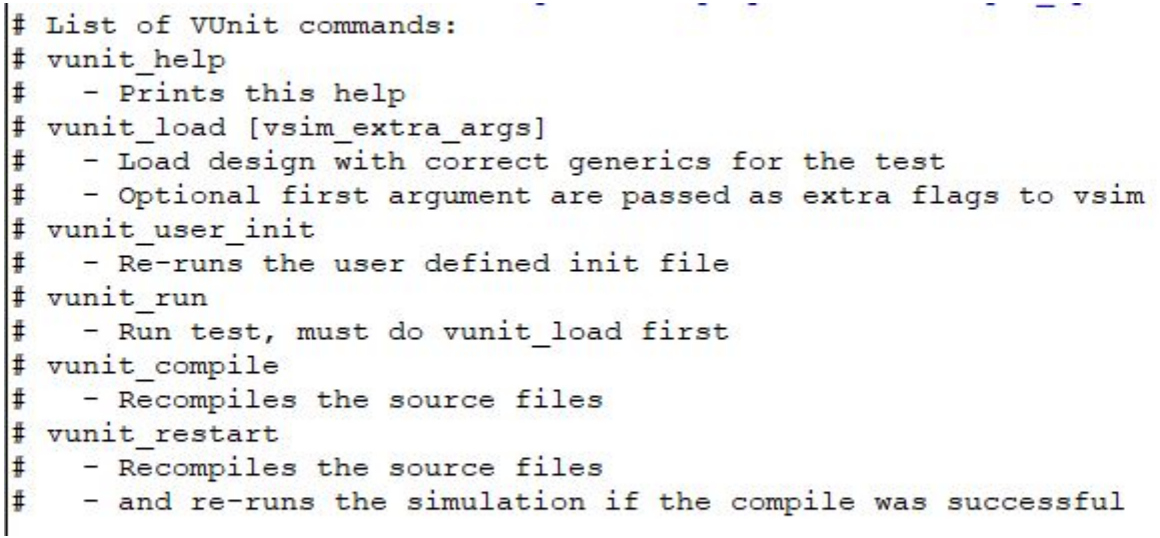
```
framework> python run.py -g (--gui)
```

The option **-g [--gui]** opens the Modelsim/Questa project and uses the

Run test bench in GUI

framework> python run.py *.send_1_byte -g

Once test bench is loaded,
VUnit’s control commands are available through
the tcl shell in Questa



```
# List of VUnit commands:
# vunit_help
#   - Prints this help
# vunit_load [vsim_extra_args]
#   - Load design with correct generics for the test
#   - Optional first argument are passed as extra flags to vsim
# vunit_user_init
#   - Re-runs the user defined init file
# vunit_run
#   - Run test, must do vunit_load first
# vunit_compile
#   - Recompiles the source files
# vunit_restart
#   - Recompiles the source files
#   - and re-runs the simulation if the compile was successful
```

Restructure testbench to a UVVM testbench

By restructuring the test bench based on UVVM's recommendations we'll be able to include [I2C, UART, Ethernet…] available BFM components into our test bench, with little changes.

Create a test harness that includes all constant and procedures existing procedures at the original test bench.

If there are constants that will be used both on the test bench and the test harness a package should be created. This package will be imported from both files in a way that they share this data.

Testbench package / i2c_controller_tb_pkg

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package i2c_controller_tb_pkg is
  -- Constant List
  constant clk_hz         : integer := 10_000_000;
  constant clk_period     : time    := 1 sec / clk_hz;
  -- I2C
  constant i2c_hz         : integer := 100_000;
  constant i2c_period     : time    :=  1 sec / i2c_hz;
  constant target_addr    : std_logic_vector(6 downto 0)  := "1010101";
end package;
```

Test harness // libraries

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

-- UVVM framework library
library uvvm_vvc_framework;
-- UVVM I2C component library
library bitvis_vip_i2c;
-- Custom VVC
library hakonix_vip_i2c_user;
-- Testbench package
use work.i2c_controller_tb_pkg.all;

entity i2c_controller_th is
end i2c_controller_th;
```

Test harness // signals

```
architecture sim of i2c_controller_th is
  signal clk        : std_logic                      := '1';
  signal rst        : std_logic                      := '1';
  -- I2C interface
  signal scl        : std_logic;
  signal sda        : std_logic;
  -- Command Bus interface // AXI
  signal cmd_tdata  : std_logic_vector(7 downto 0)   := (others => '0');
  signal cmd_tvalid : std_logic                      := '0';
  signal cmd_tready : std_logic;
  -- Read Bus interface // AXI
  signal rd_tdata   : std_logic_vector(7 downto 0);
  signal rd_tvalid  : std_logic;
  signal rd_tready  : std_logic                      := '0';
  -- Not Acknowledge // Pulsed on every received NACK
  signal nack       : std_logic;
```

Test harness

```
begin
  -- Generate clock
  clk <= not clk after clk_period / 2;
  -- Release reset
  rst <= '0' after clk_period * 2;
  -- Pullup
  scl <= 'H';
  sda <= 'H';
  -- UVVM engine module initialization is required for every UVVM testbench
  UVVM_ENGINE : entity uvvm_vvc_framework.ti_uvvm_engine(func);
```

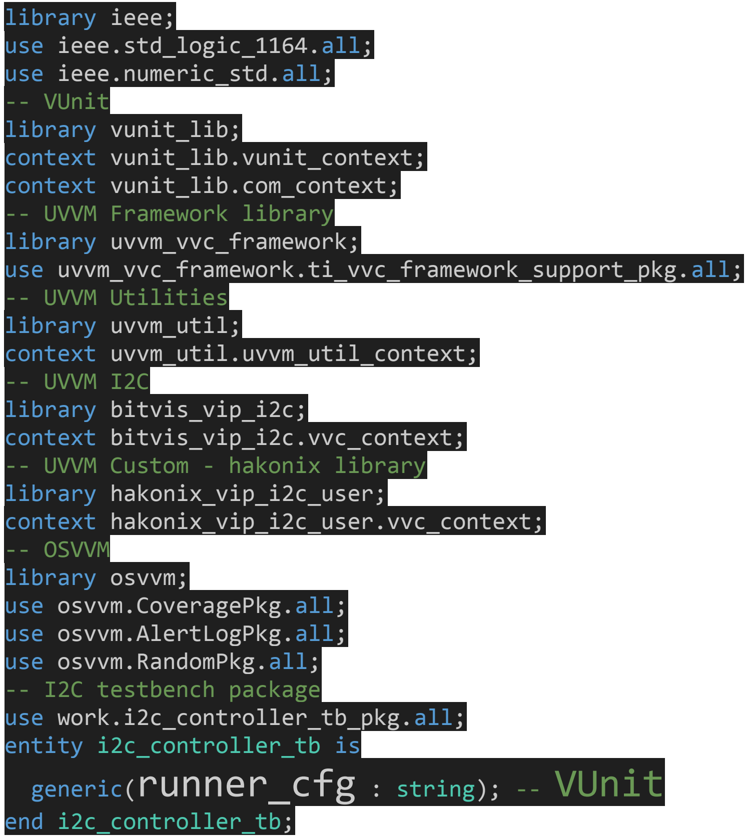
Test harness

```
DUT : entity work.i2c_controller(rtl)
generic map (
  clk_hz => clk_hz,
  i2c_hz => i2c_hz
)
port map (
  clk         => clk,
  rst         => rst,
  scl         => scl,
  sda         => sda,
  cmd_tdata   => cmd_tdata,
  cmd_tvalid  => cmd_tvalid,
  cmd_tready  => cmd_tready,
  rd_tdata    => rd_tdata,
  rd_tvalid   => rd_tvalid,
  rd_tready   => rd_tready,
  nack        => nack
);
```

Test harness

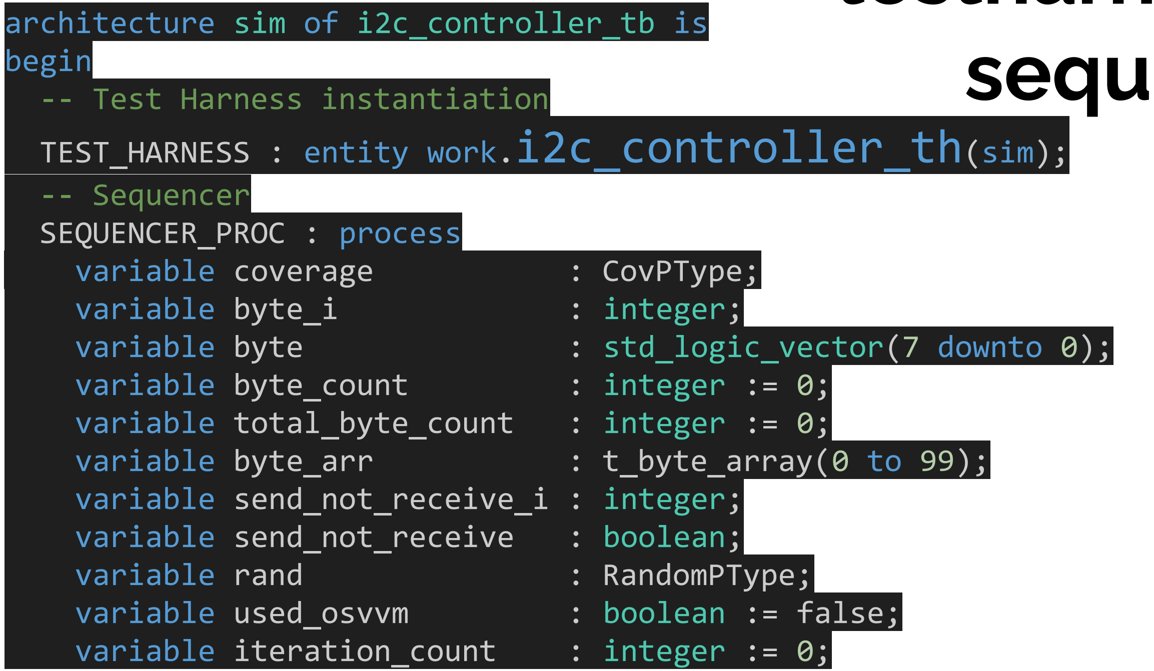
```
  I2C_VVC : entity bitvis_vip_i2c.i2c_vvc(behave)
  generic map (
    GC_MASTER_MODE  => false
  )
  port map (
    i2c_vvc_if.scl => scl,
    i2c_vvc_if.sda => sda
  );

  I2C_USER_VVC : entity hakonix_vip_i2c_user.i2c_user_vvc(behave)
    port map (
    clk                        => clk,
    i2c_user_vvc_if.cmd_tdata  => cmd_tdata,  -- to dut
    i2c_user_vvc_if.cmd_tvalid => cmd_tvalid, -- to dut
    i2c_user_vvc_if.cmd_tready => cmd_tready, -- from dut
    i2c_user_vvc_if.rd_tdata   => rd_tdata,   -- from dut
    i2c_user_vvc_if.rd_tvalid  => rd_tvalid,  -- from dut
    i2c_user_vvc_if.rd_tready  => rd_tready,  -- to dut
    i2c_user_vvc_if.nack       => nack        -- from dut
    );

end architecture;
```

Testbench // libraries



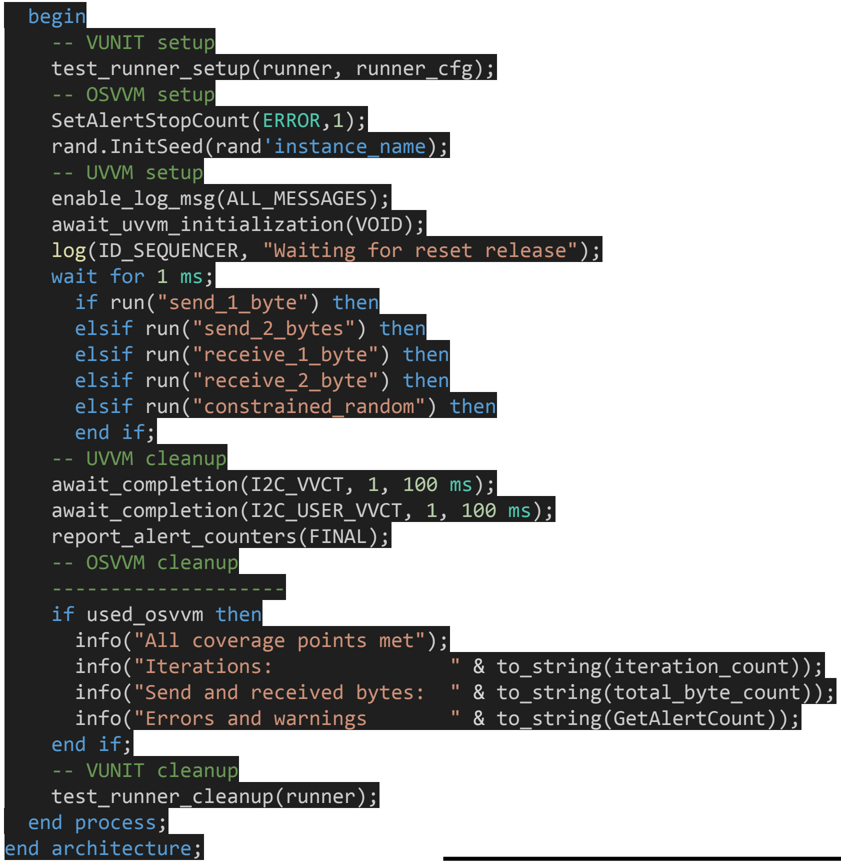
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-- VUnit
library vunit_lib;
context vunit_lib.vunit_context;
context vunit_lib.com_context;
-- UVVM Framework library
library uvvm_vvc_framework;
use uvvm_vvc_framework.ti_vvc_framework_support_pkg.all;
-- UVVM Utilities
library uvvm_util;
context uvvm_util.uvvm_util_context;
-- UVVM I2C
library bitvis_vip_i2c;
context bitvis_vip_i2c.vvc_context;
-- UVVM Custom - hakonix library
library hakonix_vip_i2c_user;
context hakonix_vip_i2c_user.vvc_context;
-- OSVVM
library osvvm;
use osvvm.CoveragePkg.all;
use osvvm.AlertLogPkg.all;
use osvvm.RandomPkg.all;
-- I2C testbench package
use work.i2c_controller_tb_pkg.all;
entity i2c_controller_tb is
  generic(runner_cfg : string); -- VUnit
end i2c_controller_tb;
```

Testbench // testharness instance // sequencer variables



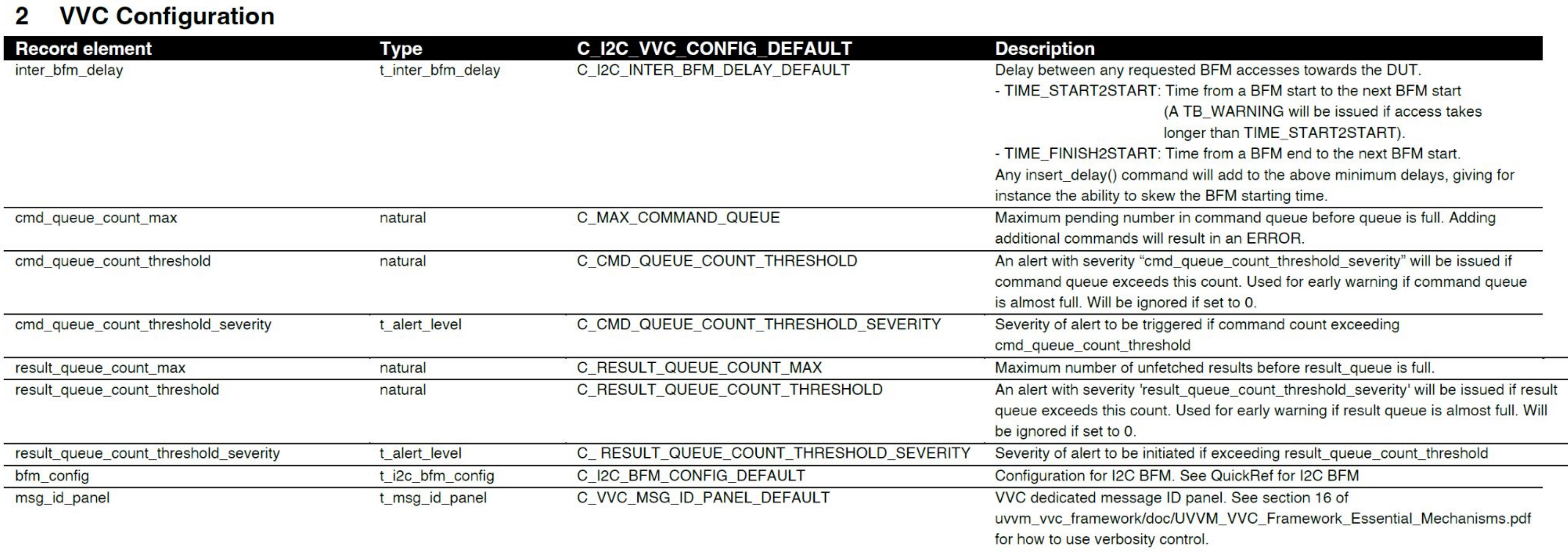
```
architecture sim of i2c_controller_tb is
begin
  -- Test Harness instantiation
  TEST_HARNESS : entity work.i2c_controller_th(sim);
  -- Sequencer
  SEQUENCER_PROC : process
    variable coverage           : CovPType;
    variable byte_i             : integer;
    variable byte               : std_logic_vector(7 downto 0);
    variable byte_count         : integer := 0;
    variable total_byte_count   : integer := 0;
    variable byte_arr           : t_byte_array(0 to 99);
    variable send_not_receive_i : integer;
    variable send_not_receive   : boolean;
    variable rand               : RandomPType;
    variable used_osvvm         : boolean := false;
    variable iteration_count    : integer := 0;
```

Testbench // sequencer profile



```
  begin
    -- VUNIT setup
    test_runner_setup(runner, runner_cfg);
    -- OSVVM setup
    SetAlertStopCount(ERROR,1);
    rand.InitSeed(rand'instance_name);
    -- UVVM setup
    enable_log_msg(ALL_MESSAGES);
    await_uvvm_initialization(VOID);
    log(ID_SEQUENCER, "Waiting for reset release");
    wait for 1 ms;
      if run("send_1_byte") then
      elsif run("send_2_bytes") then
      elsif run("receive_1_byte") then
      elsif run("receive_2_byte") then
      elsif run("constrained_random") then
      end if;
    -- UVVM cleanup
    await_completion(I2C_VVCT, 1, 100 ms);
    await_completion(I2C_USER_VVCT, 1, 100 ms);
    report_alert_counters(FINAL);
    -- OSVVM cleanup
    --------------------
    if used_osvvm then
      info("All coverage points met");
      info("Iterations:               " & to_string(iteration_count));
      info("Send and received bytes:  " & to_string(total_byte_count));
      info("Errors and warnings       " & to_string(GetAlertCount));
    end if;
    -- VUNIT cleanup
    test_runner_cleanup(runner);
  end process;
end architecture;
```

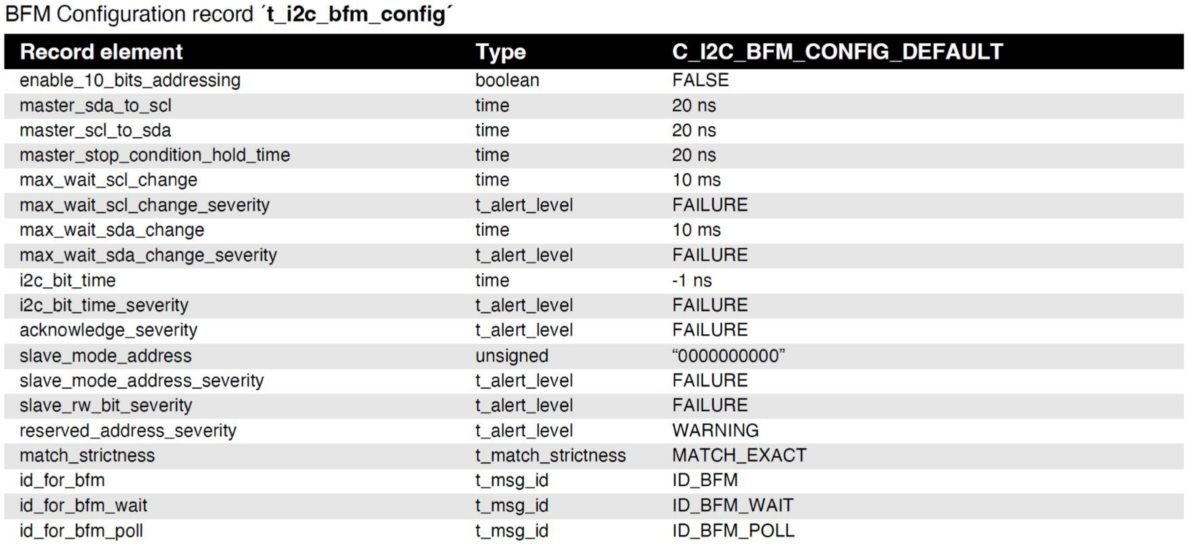
I2C VVC - bfm_config (t_i2c_bfm_config)



2 VVC Configuration

Record element	Type	C_I2C_VVC_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_I2C_INTER_BFM_DELAY_DEFAULT	Delay between any requested BFM accesses towards the DUT. - TIME_START2START: Time from a BFM start to the next BFM start (A TB_WARNING will be issued if access takes longer than TIME_START2START). - TIME_FINISH2START: Time from a BFM end to the next BFM start. Any insert_delay() command will add to the above minimum delays, giving for instance the ability to skew the BFM starting time.
cmd_queue_count_max	natural	C_MAX_COMMAND_QUEUE	Maximum pending number in command queue before queue is full. Adding additional commands will result in an ERROR.
cmd_queue_count_threshold	natural	C_CMD_QUEUE_COUNT_THRESHOLD	An alert with severity “cmd_queue_count_threshold_severity” will be issued if command queue exceeds this count. Used for early warning if command queue is almost full. Will be ignored if set to 0.
cmd_queue_count_threshold_severity	t_alert_level	C_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be triggered if command count exceeding cmd_queue_count_threshold
result_queue_count_max	natural	C_RESULT_QUEUE_COUNT_MAX	Maximum number of unfetched results before result_queue is full.
result_queue_count_threshold	natural	C_RESULT_QUEUE_COUNT_THRESHOLD	An alert with severity 'result_queue_count_threshold_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0.
result_queue_count_threshold_severity	t_alert_level	C_ RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be initiated if exceeding result_queue_count_threshold
bfm_config	t_i2c_bfm_config	C_I2C_BFM_CONFIG_DEFAULT	Configuration for I2C BFM. See QuickRef for I2C BFM
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT	VVC dedicated message ID panel. See section 16 of uvvm_vvc_framework/doc/UVVM_VVC_Framework_Essential_Mechanisms.pdf for how to use verbosity control.

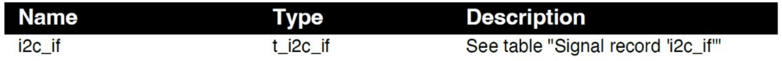
I2C BFM - C_I2C_BFM_CONFIG_DEFAULT



BFM Configuration record ´**t_i2c_bfm_config**´

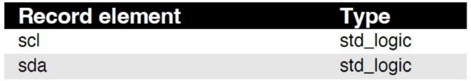
Record element	Type	C_I2C_BFM_CONFIG_DEFAULT
enable_10_bits_addressing	boolean	FALSE
master_sda_to_scl	time	20 ns
master_scl_to_sda	time	20 ns
master_stop_condition_hold_time	time	20 ns
max_wait_scl_change	time	10 ms
max_wait_scl_change_severity	t_alert_level	FAILURE
max_wait_sda_change	time	10 ms
max_wait_sda_change_severity	t_alert_level	FAILURE
i2c_bit_time	time	-1 ns
i2c_bit_time_severity	t_alert_level	FAILURE
acknowledge_severity	t_alert_level	FAILURE
slave_mode_address	unsigned	“0000000000”
slave_mode_address_severity	t_alert_level	FAILURE
slave_rw_bit_severity	t_alert_level	FAILURE
reserved_address_severity	t_alert_level	WARNING
match_strictness	t_match_strictness	MATCH_EXACT
id_for_bfm	t_msg_id	ID_BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT
id_for_bfm_poll	t_msg_id	ID_BFM_POLL

BFM signal parameters



Name	Type	Description
i2c_if	t_i2c_if	See table "Signal record 'i2c_if'"

Signal record ´t_i2c_if´



Record element	Type
scl	std_logic
sda	std_logic

UVVM’s I2C slave & i2C user config

Include the library on the test bench

```
-- UVVM I2C
library bitvis_vip_i2c;
context bitvis_vip_i2c.vvc_context;
-- UVVM Custom - hakonix library
library hakonix_vip_i2c_user;
context hakonix_vip_i2c_user.vvc_context;
```

Assign the values to the share variables used for configuring VVC components

```
    -- UVVM I2C VVC configuration
    shared_i2c_vvc_config(1).bfm_config.master_sda_to_scl               := i2c_period;
    shared_i2c_vvc_config(1).bfm_config.master_scl_to_sda               := i2c_period;
    shared_i2c_vvc_config(1).bfm_config.max_wait_scl_change             := i2c_period;
    shared_i2c_vvc_config(1).bfm_config.max_wait_sda_change             := i2c_period;
    shared_i2c_vvc_config(1).bfm_config.i2c_bit_time                    := i2c_period;
    shared_i2c_vvc_config(1).bfm_config.slave_mode_address(6 downto 0)  := unsigned(target_addr);
    -- Hakonix I2C VVC configuration
    shared_i2c_user_vvc_config(1).bfm_config.bit_period                 := i2c_period;
    shared_i2c_user_vvc_config(1).bfm_config.target_addr                := target_addr;
```

Custom UVVM verification component

Run the script as it follows; an output folder will be created, rename it **hakonix_vip_i2c_user**

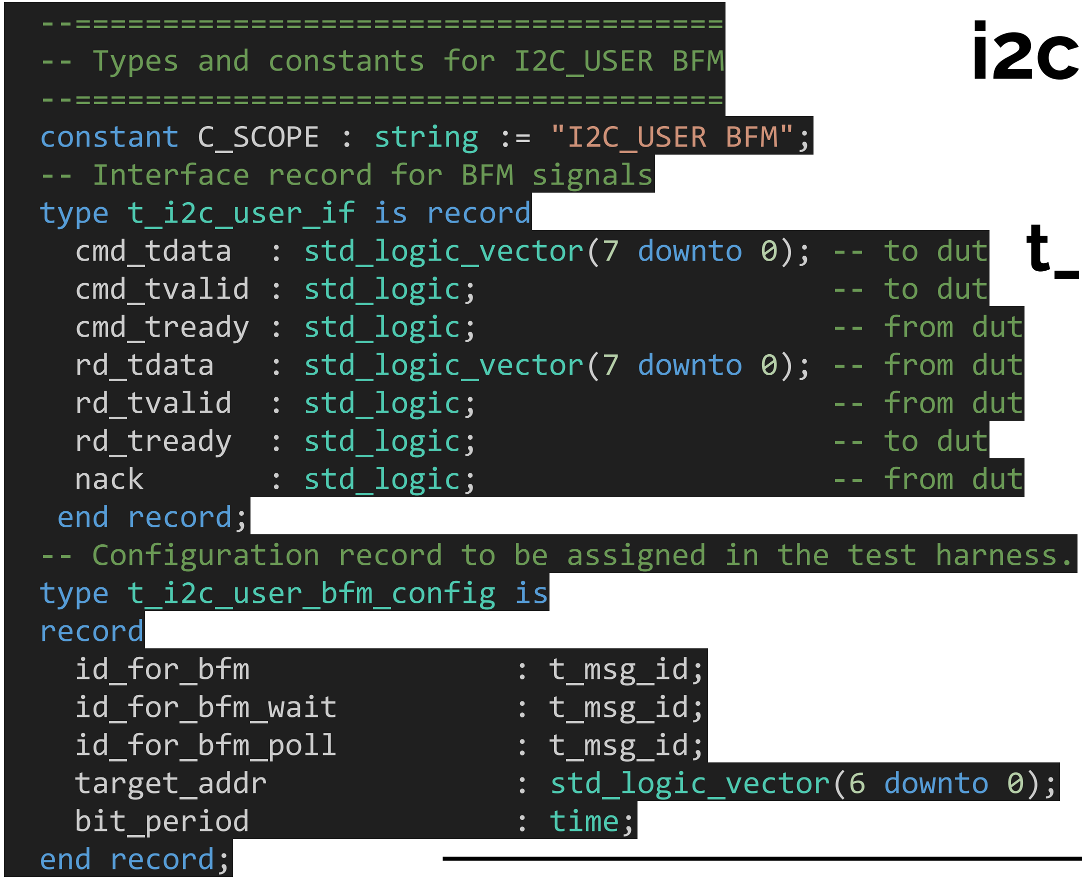
```
framewokr_proj> python
.\uvvm\uvvm_vvc_framework\script\vvc_generator\vvc_generator.py
Please enter the VVC Name (e.g. SBI, UART, axilite): i2c_user

The VVC is generated with basic code for running with UVVM as default, but can be
generated with extended UVVM features such as Scoreboard and transaction info.
Generate VVC with extended UVVM features? [y/n]: n

Multiple channels can be used to emulate concurrent channels in the VIP, e.g. concurrent RX and
TX channels.
Set the number of concurrent channels to use [1-99], press enter for default(1):

Multiple executors (and queues) are used when concurrent command operations are needed.

Shall the VVC have multiple executors? [y/n]: n

The vvc_generator script is now finished
```

vvc_context

Update the references on the context

```
context vvc_context is
  library hakonix_vip_i2c_user;
  use hakonix_vip_i2c_user.vvc_methods_pkg.all;
  use hakonix_vip_i2c_user.td_vvc_framework_common_methods_pkg.all;
  use hakonix_vip_i2c_user.i2c_user_bfm_pkg.all;
end context;
```

i2c_user_bfm_pkg / t_i2c_user_if + t_i2c_user_bfm_config



```vhdl
--=====================================
-- Types and constants for I2C_USER BFM
--=====================================
constant C_SCOPE : string := "I2C_USER BFM";
-- Interface record for BFM signals
type t_i2c_user_if is record
  cmd_tdata   : std_logic_vector(7 downto 0); -- to dut
  cmd_tvalid  : std_logic;                    -- to dut
  cmd_tready  : std_logic;                    -- from dut
  rd_tdata    : std_logic_vector(7 downto 0); -- from dut
  rd_tvalid   : std_logic;                    -- from dut
  rd_tready   : std_logic;                    -- to dut
  nack        : std_logic;                    -- from dut
 end record;
-- Configuration record to be assigned in the test harness.
type t_i2c_user_bfm_config is
record
  id_for_bfm                : t_msg_id;
  id_for_bfm_wait           : t_msg_id;
  id_for_bfm_poll           : t_msg_id;
  target_addr               : std_logic_vector(6 downto 0);
  bit_period                : time;
end record;
```

i2c_user_bfm_pkg / C_I2C_USER_BFM_CONFIG_DEFAULT

```
  -- Define the default value for the BFM config
  constant C_I2C_USER_BFM_CONFIG_DEFAULT : t_i2c_user_bfm_config := (
    id_for_bfm                 => ID_BFM,
    id_for_bfm_wait            => ID_BFM_WAIT,
    id_for_bfm_poll            => ID_BFM_POLL,
    target_addr                => “0000000”,
    bit_period                 => -1 ns,
  );
```

i2c_user_bfm_pkg /init_i2c_user_if_signals function

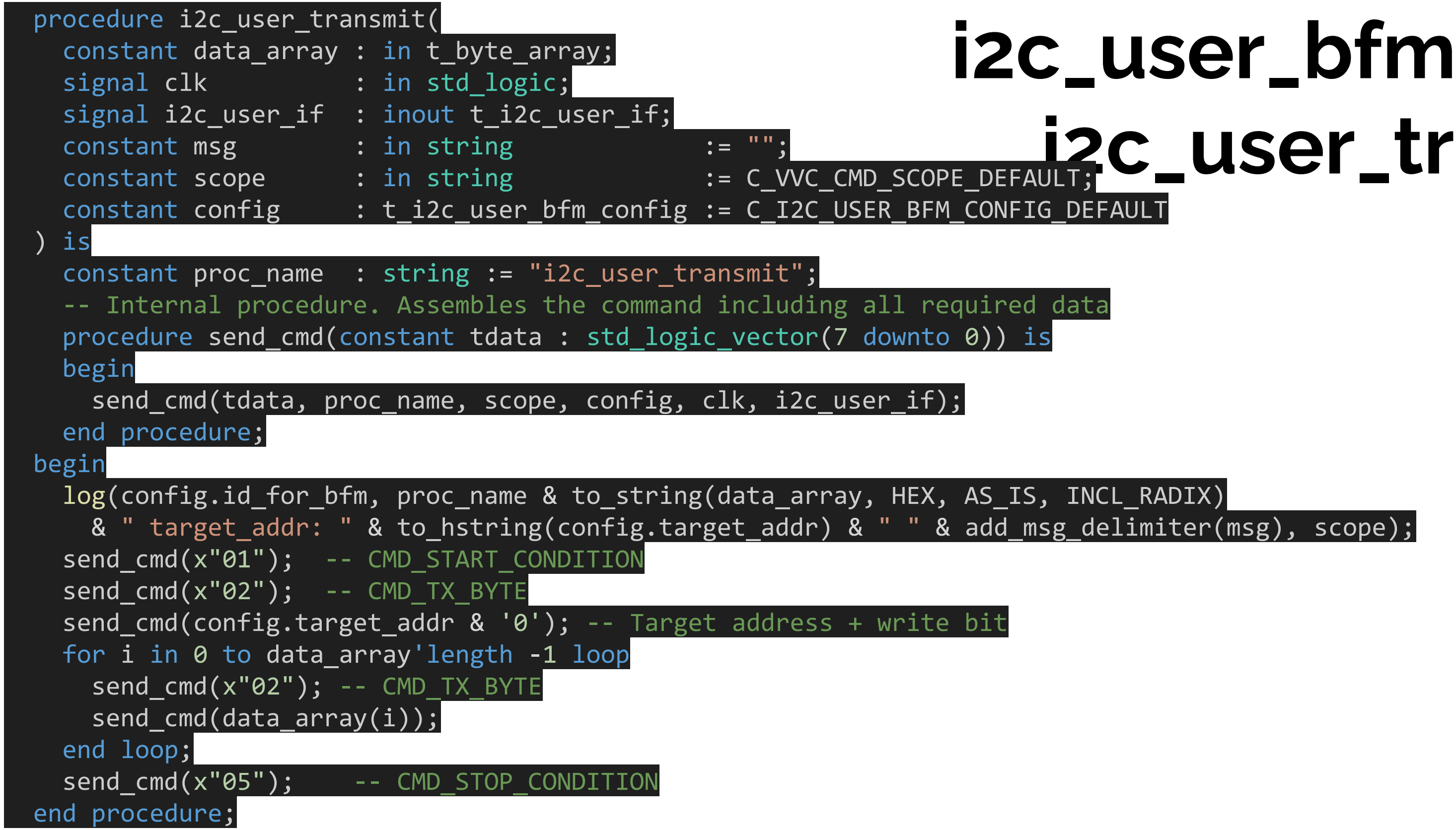
```vhdl
  function init_i2c_user_if_signals return t_i2c_user_if is
    variable r : t_i2c_user_if;
  begin
    -- Initialize all elements of type T_I2C_USER_IF
    r.cmd_tdata   := (others => 'X');
    r.cmd_tvalid  := '0';
    r.cmd_tready  := 'Z';
    r.rd_tdata    := (others => 'Z');
    r.rd_tvalid   := 'Z';
    r.rd_tready   := '0';
    r.nack        := 'Z';
    -- Return initialized values
    return r;
  end function;
```

Declare the prototype and then implement the initialization in the package body as shown above

I2c_user_bfm_pkg / send_cmd

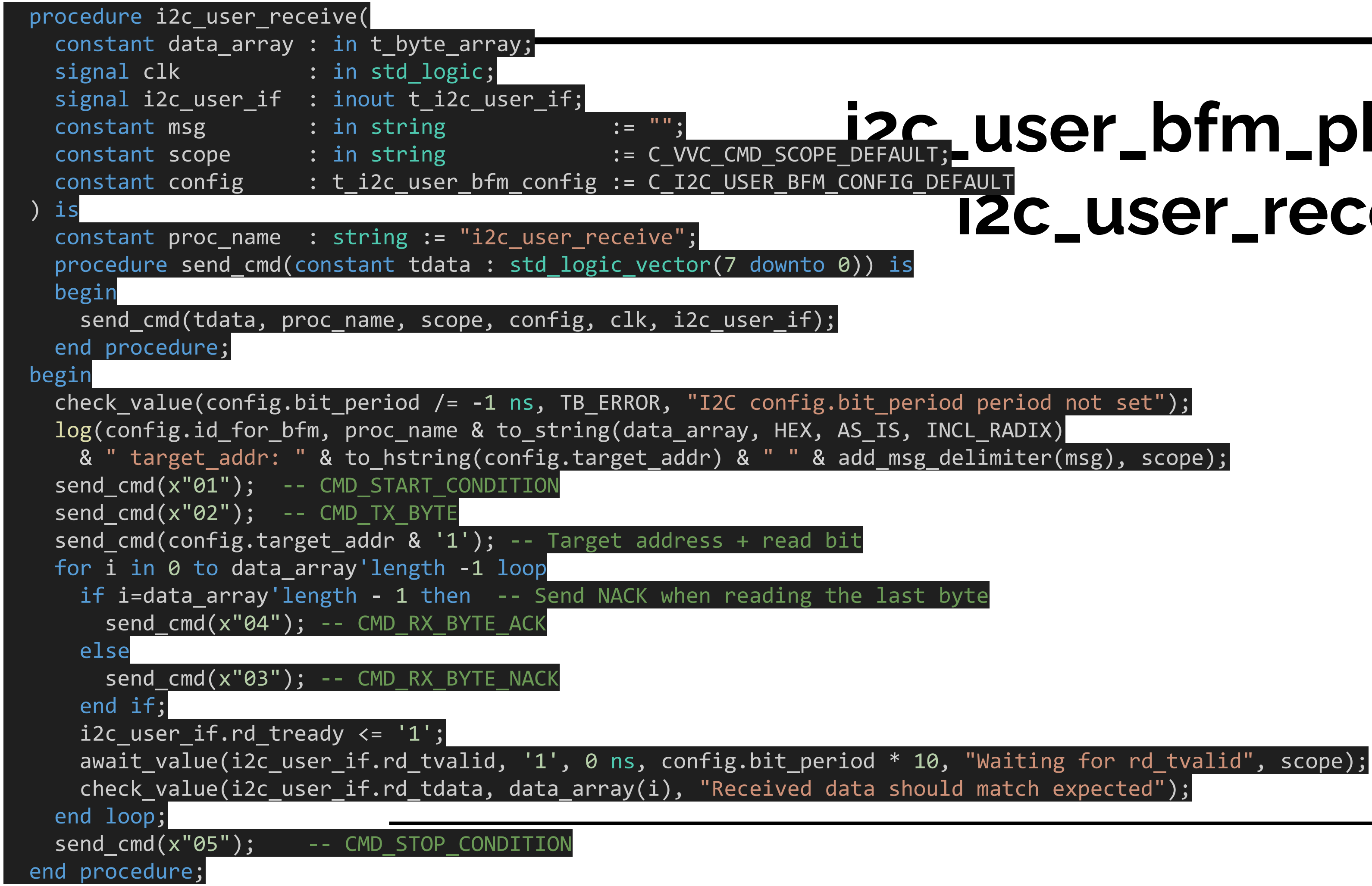
```vhdl
-- Send a command to the I2C controller
procedure send_cmd(
  constant tdata        : std_logic_vector(7 downto 0);
  constant proc_name    : string;
  constant scope        : string;
  constant config       : t_i2c_user_bfm_config;
  signal   clk          : std_logic;
  signal   i2c_user_if  : inout t_i2c_user_if
  ) is
begin
  log(config.id_for_bfm, proc_name & "(): receive cmd byte: 0x" & to_hstring(tdata), scope);
  i2c_user_if.cmd_tdata   <= tdata;
  i2c_user_if.cmd_tvalid  <= '1';
  loop
    wait until rising_edge(clk);
    if i2c_user_if.cmd_tready = '1' then
      exit;
    end if;
  end loop;
  i2c_user_if.cmd_tdata   <= (others => 'X');
  i2c_user_if.cmd_tvalid  <= '0';
end procedure;
```

i2c_user_bfm_pkg / i2c_user_transmit



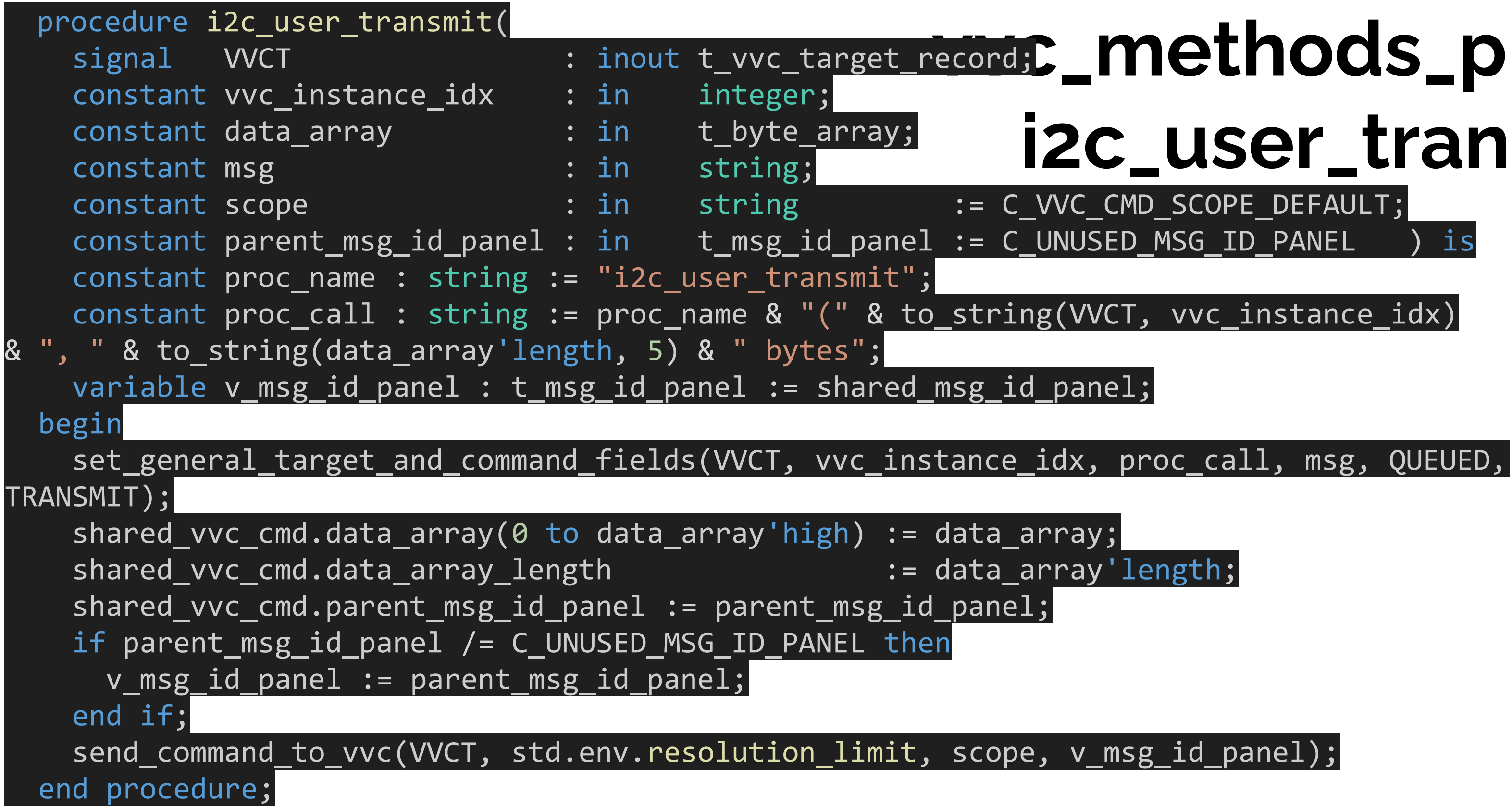
```
procedure i2c_user_transmit(
  constant data_array : in t_byte_array;
  signal clk          : in std_logic;
  signal i2c_user_if  : inout t_i2c_user_if;
  constant msg        : in string                := "";
  constant scope      : in string                := C_VVC_CMD_SCOPE_DEFAULT;
  constant config     : t_i2c_user_bfm_config := C_I2C_USER_BFM_CONFIG_DEFAULT
) is
  constant proc_name  : string := "i2c_user_transmit";
  -- Internal procedure. Assembles the command including all required data
  procedure send_cmd(constant tdata : std_logic_vector(7 downto 0)) is
  begin
    send_cmd(tdata, proc_name, scope, config, clk, i2c_user_if);
  end procedure;
begin
  log(config.id_for_bfm, proc_name & to_string(data_array, HEX, AS_IS, INCL_RADIX)
    & " target_addr: " & to_hstring(config.target_addr) & " " & add_msg_delimiter(msg), scope);
  send_cmd(x"01");  -- CMD_START_CONDITION
  send_cmd(x"02");  -- CMD_TX_BYTE
  send_cmd(config.target_addr & '0'); -- Target address + write bit
  for i in 0 to data_array'length -1 loop
    send_cmd(x"02"); -- CMD_TX_BYTE
    send_cmd(data_array(i));
  end loop;
  send_cmd(x"05");    -- CMD_STOP_CONDITION
end procedure;
```

i2c_user_bfm_pkg / i2c_user_receive



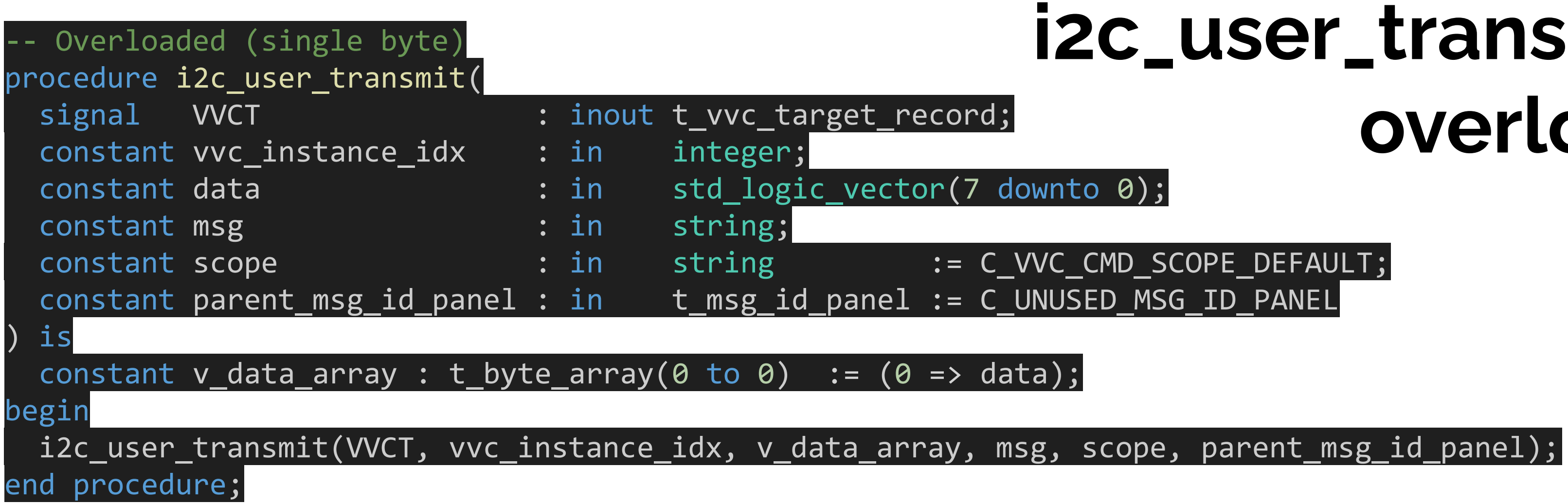
```vhdl
procedure i2c_user_receive(
  constant data_array : in t_byte_array;
  signal clk          : in std_logic;
  signal i2c_user_if  : inout t_i2c_user_if;
  constant msg        : in string                := "";
  constant scope      : in string                := C_VVC_CMD_SCOPE_DEFAULT;
  constant config     : t_i2c_user_bfm_config := C_I2C_USER_BFM_CONFIG_DEFAULT
) is
  constant proc_name  : string := "i2c_user_receive";
  procedure send_cmd(constant tdata : std_logic_vector(7 downto 0)) is
  begin
    send_cmd(tdata, proc_name, scope, config, clk, i2c_user_if);
  end procedure;
begin
  check_value(config.bit_period /= -1 ns, TB_ERROR, "I2C config.bit_period period not set");
  log(config.id_for_bfm, proc_name & to_string(data_array, HEX, AS_IS, INCL_RADIX)
    & " target_addr: " & to_hstring(config.target_addr) & " " & add_msg_delimiter(msg), scope);
  send_cmd(x"01");  -- CMD_START_CONDITION
  send_cmd(x"02");  -- CMD_TX_BYTE
  send_cmd(config.target_addr & '1'); -- Target address + read bit
  for i in 0 to data_array'length -1 loop
    if i=data_array'length - 1 then  -- Send NACK when reading the last byte
      send_cmd(x"04"); -- CMD_RX_BYTE_ACK
    else
      send_cmd(x"03"); -- CMD_RX_BYTE_NACK
    end if;
    i2c_user_if.rd_tready <= '1';
    await_value(i2c_user_if.rd_tvalid, '1', 0 ns, config.bit_period * 10, "Waiting for rd_tvalid", scope);
    check_value(i2c_user_if.rd_tdata, data_array(i), "Received data should match expected");
  end loop;
  send_cmd(x"05");    -- CMD_STOP_CONDITION
end procedure;
```

vvc_methods_pkg / i2c_user_transmit



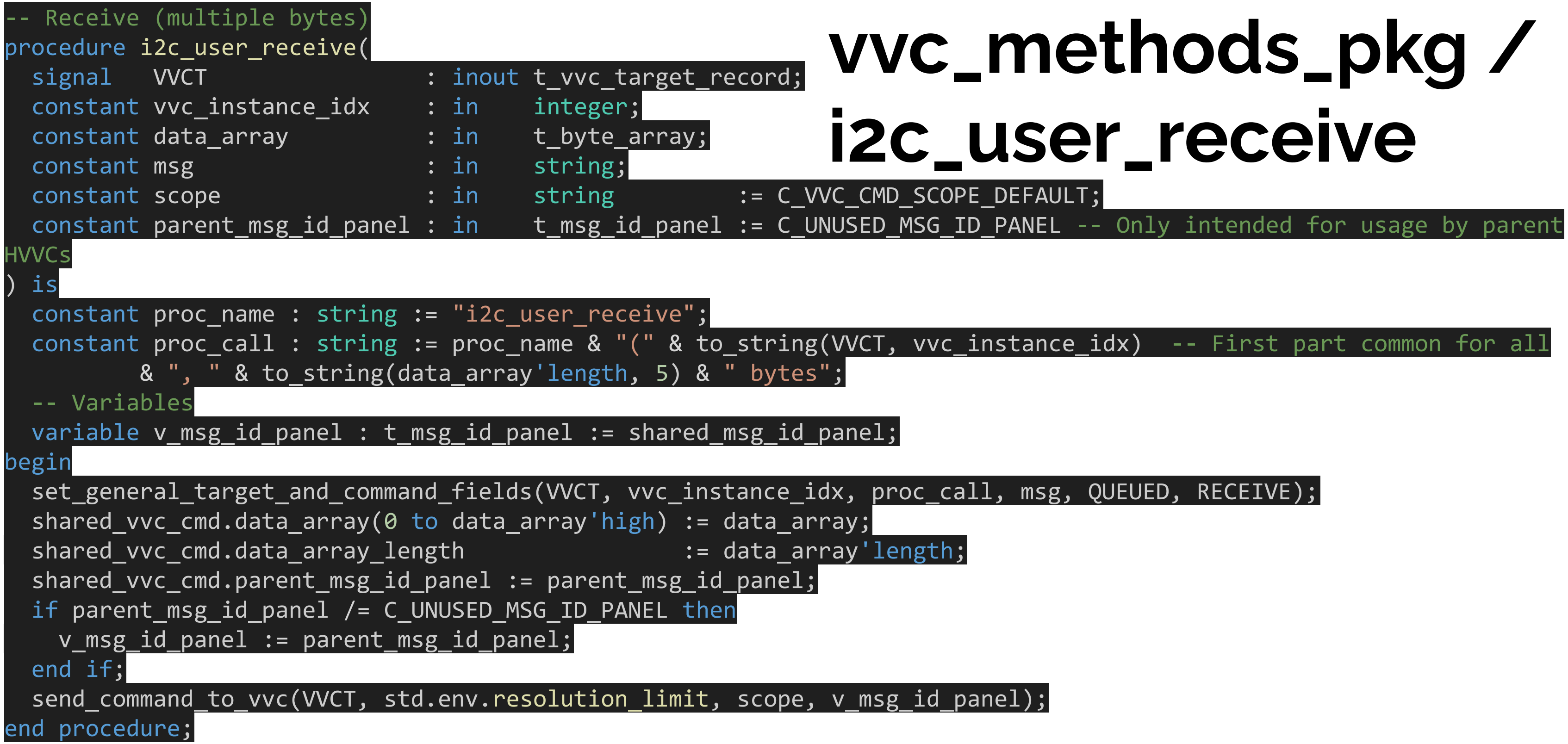
```
  procedure i2c_user_transmit(
    signal   VVCT                  : inout t_vvc_target_record;
    constant vvc_instance_idx      : in    integer;
    constant data_array            : in    t_byte_array;
    constant msg                   : in    string;
    constant scope                 : in    string          := C_VVC_CMD_SCOPE_DEFAULT;
    constant parent_msg_id_panel : in    t_msg_id_panel := C_UNUSED_MSG_ID_PANEL   ) is
    constant proc_name : string := "i2c_user_transmit";
    constant proc_call : string := proc_name & "(" & to_string(VVCT, vvc_instance_idx)
& ", " & to_string(data_array'length, 5) & " bytes";
    variable v_msg_id_panel : t_msg_id_panel := shared_msg_id_panel;
  begin
    set_general_target_and_command_fields(VVCT, vvc_instance_idx, proc_call, msg, QUEUED,
TRANSMIT);
    shared_vvc_cmd.data_array(0 to data_array'high) := data_array;
    shared_vvc_cmd.data_array_length                := data_array'length;
    shared_vvc_cmd.parent_msg_id_panel := parent_msg_id_panel;
    if parent_msg_id_panel /= C_UNUSED_MSG_ID_PANEL then
      v_msg_id_panel := parent_msg_id_panel;
    end if;
    send_command_to_vvc(VVCT, std.env.resolution_limit, scope, v_msg_id_panel);
  end procedure;
```

vvc_methods_pkg / i2c_user_transmit overload



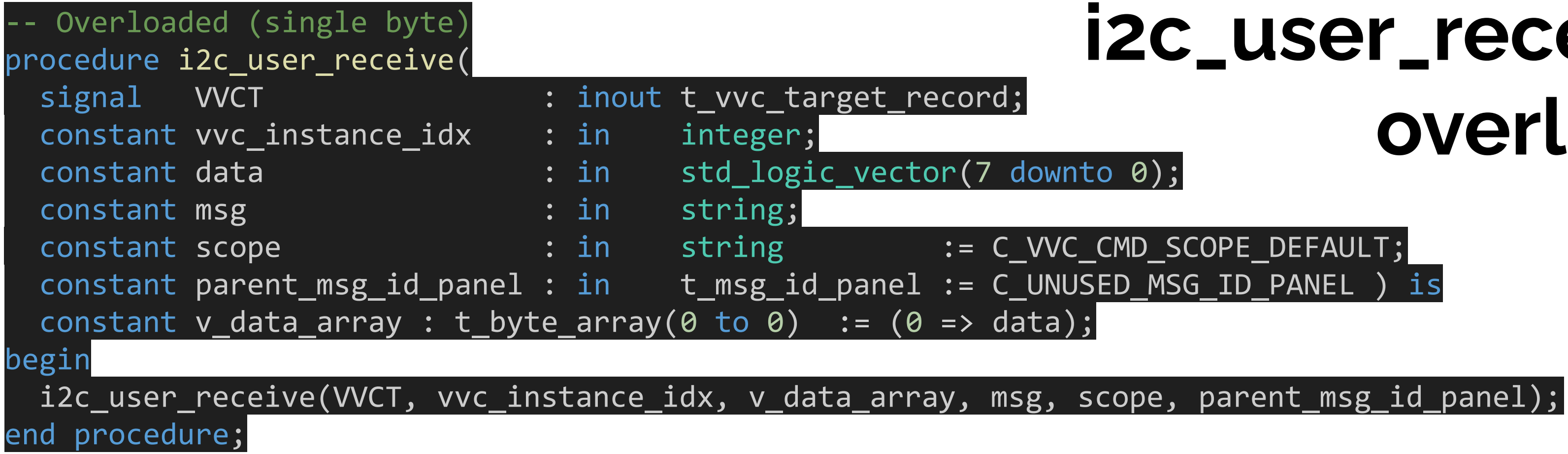
```
-- Overloaded (single byte)
procedure i2c_user_transmit(
  signal   VVCT                 : inout t_vvc_target_record;
  constant vvc_instance_idx     : in    integer;
  constant data                 : in    std_logic_vector(7 downto 0);
  constant msg                  : in    string;
  constant scope                : in    string          := C_VVC_CMD_SCOPE_DEFAULT;
  constant parent_msg_id_panel  : in    t_msg_id_panel  := C_UNUSED_MSG_ID_PANEL
) is
  constant v_data_array : t_byte_array(0 to 0)  := (0 => data);
begin
  i2c_user_transmit(VVCT, vvc_instance_idx, v_data_array, msg, scope, parent_msg_id_panel);
end procedure;
```

vvc_methods_pkg / i2c_user_receive



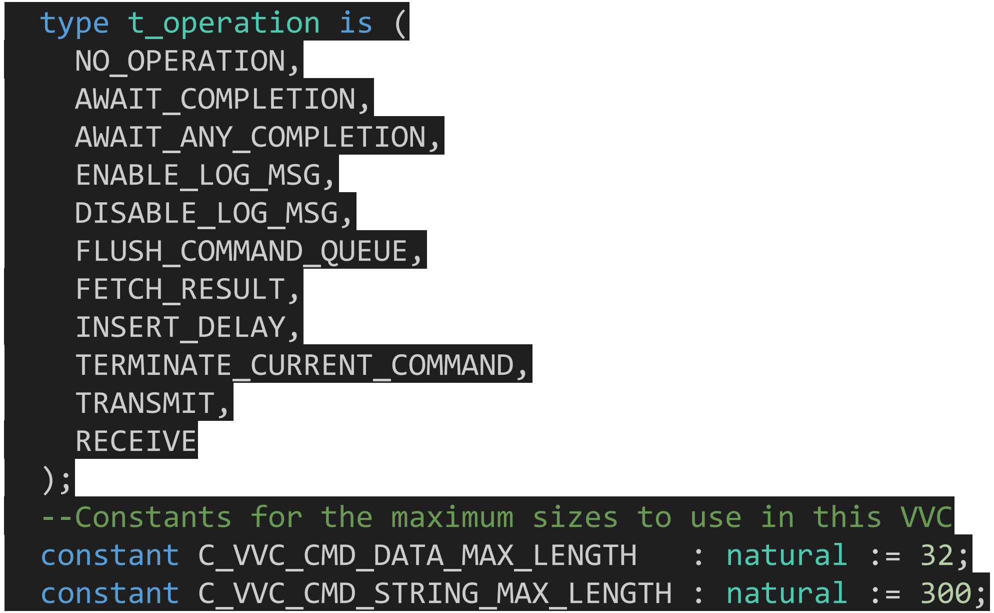
```
-- Receive (multiple bytes)
procedure i2c_user_receive(
  signal   VVCT                 : inout t_vvc_target_record;
  constant vvc_instance_idx     : in    integer;
  constant data_array           : in    t_byte_array;
  constant msg                  : in    string;
  constant scope                : in    string          := C_VVC_CMD_SCOPE_DEFAULT;
  constant parent_msg_id_panel : in    t_msg_id_panel := C_UNUSED_MSG_ID_PANEL -- Only intended for usage by parent
HVVCs
) is
  constant proc_name : string := "i2c_user_receive";
  constant proc_call : string := proc_name & "(" & to_string(VVCT, vvc_instance_idx)  -- First part common for all
          & ", " & to_string(data_array'length, 5) & " bytes";
  -- Variables
  variable v_msg_id_panel : t_msg_id_panel := shared_msg_id_panel;
begin
  set_general_target_and_command_fields(VVCT, vvc_instance_idx, proc_call, msg, QUEUED, RECEIVE);
  shared_vvc_cmd.data_array(0 to data_array'high) := data_array;
  shared_vvc_cmd.data_array_length                := data_array'length;
  shared_vvc_cmd.parent_msg_id_panel := parent_msg_id_panel;
  if parent_msg_id_panel /= C_UNUSED_MSG_ID_PANEL then
    v_msg_id_panel := parent_msg_id_panel;
  end if;
  send_command_to_vvc(VVCT, std.env.resolution_limit, scope, v_msg_id_panel);
end procedure;
```

vvc_methods_pkg / i2c_user_receive overload



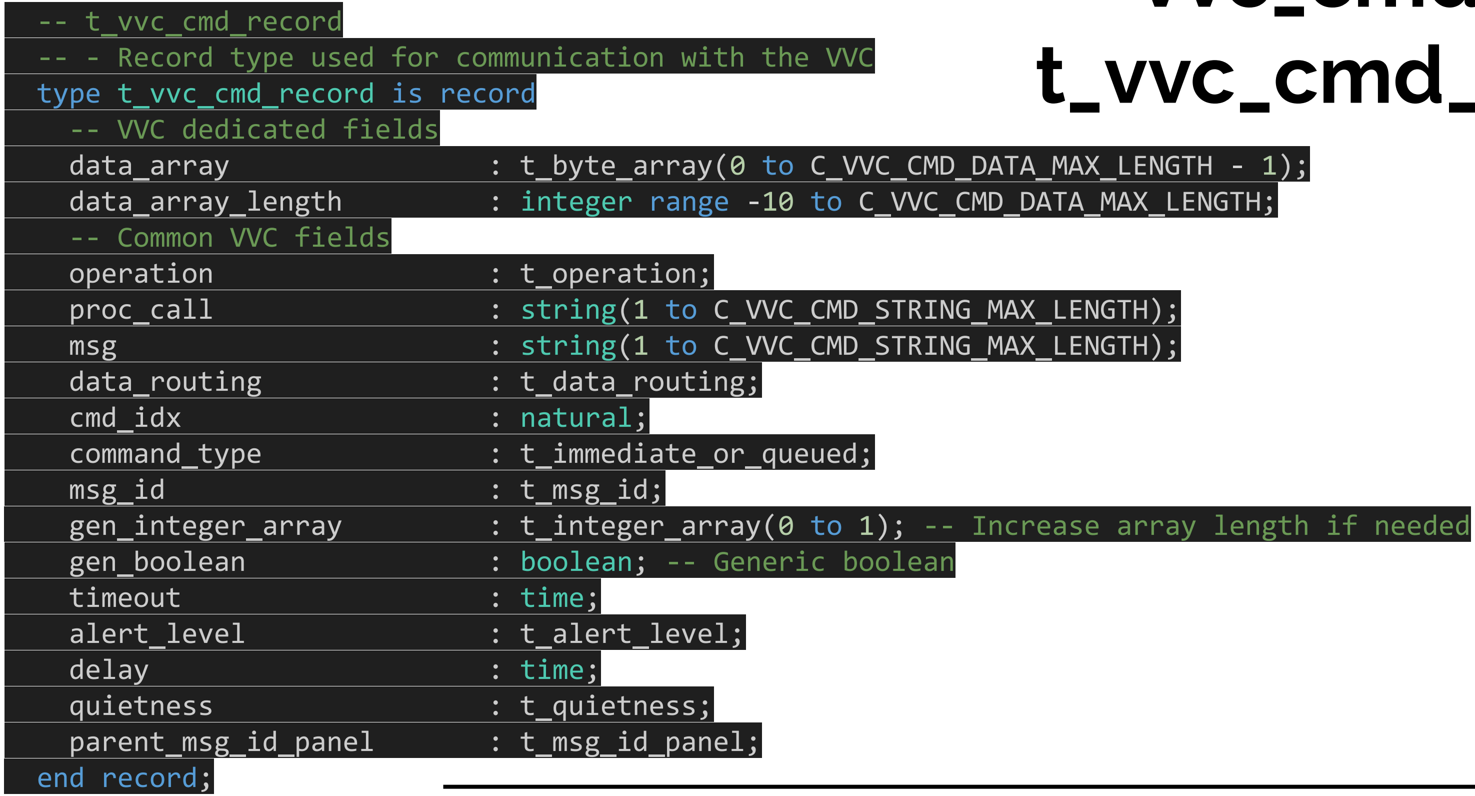
```
-- Overloaded (single byte)
procedure i2c_user_receive(
  signal   VVCT                : inout t_vvc_target_record;
  constant vvc_instance_idx    : in    integer;
  constant data                : in    std_logic_vector(7 downto 0);
  constant msg                 : in    string;
  constant scope               : in    string         := C_VVC_CMD_SCOPE_DEFAULT;
  constant parent_msg_id_panel : in    t_msg_id_panel := C_UNUSED_MSG_ID_PANEL ) is
  constant v_data_array : t_byte_array(0 to 0)  := (0 => data);
begin
  i2c_user_receive(VVCT, vvc_instance_idx, v_data_array, msg, scope, parent_msg_id_panel);
end procedure;
```

vvc_cmd_pkg / t_operation / constants



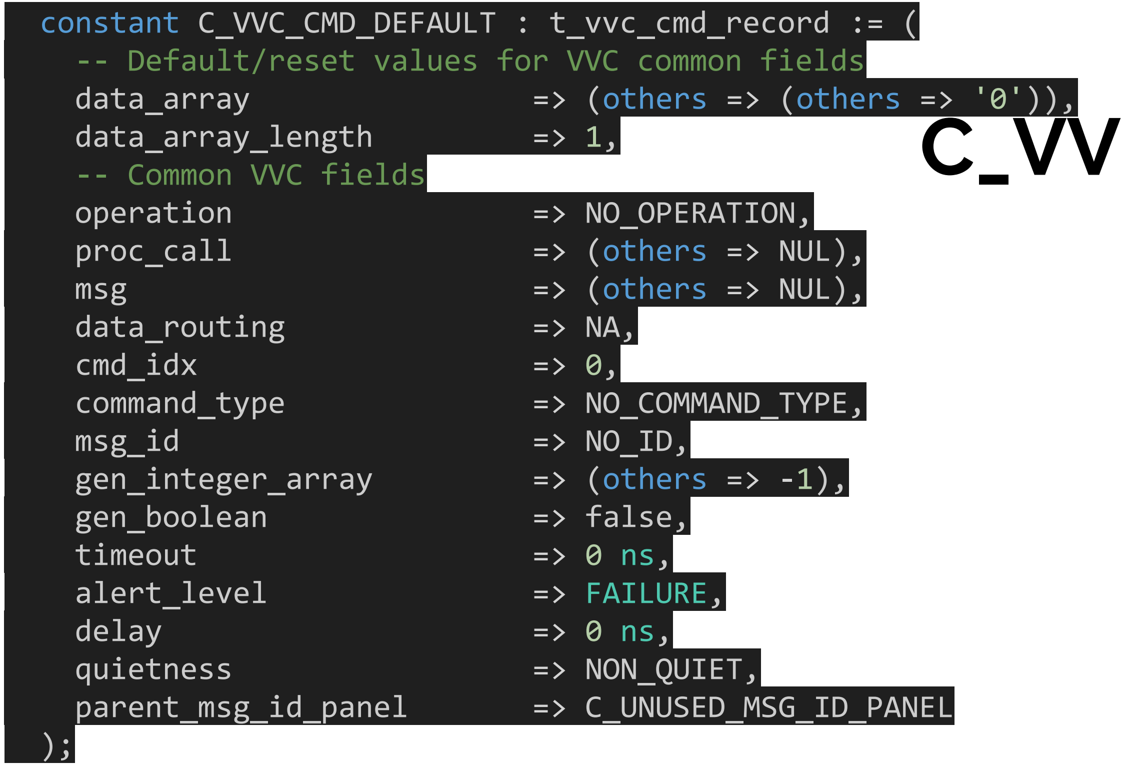
```
type t_operation is (
  NO_OPERATION,
  AWAIT_COMPLETION,
  AWAIT_ANY_COMPLETION,
  ENABLE_LOG_MSG,
  DISABLE_LOG_MSG,
  FLUSH_COMMAND_QUEUE,
  FETCH_RESULT,
  INSERT_DELAY,
  TERMINATE_CURRENT_COMMAND,
  TRANSMIT,
  RECEIVE
);
--Constants for the maximum sizes to use in this VVC
constant C_VVC_CMD_DATA_MAX_LENGTH   : natural := 32;
constant C_VVC_CMD_STRING_MAX_LENGTH : natural := 300;
```

vvc_cmd_pkg / t_vvc_cmd_record



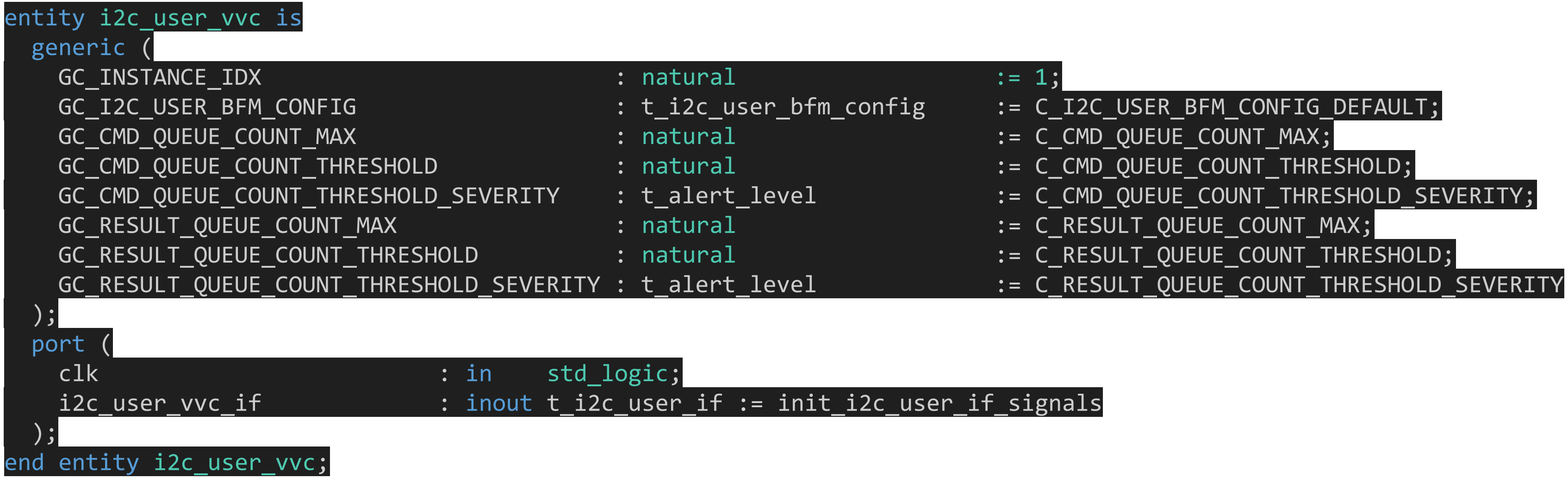
```
-- t_vvc_cmd_record
-- - Record type used for communication with the VVC
type t_vvc_cmd_record is record
  -- VVC dedicated fields
  data_array                : t_byte_array(0 to C_VVC_CMD_DATA_MAX_LENGTH - 1);
  data_array_length         : integer range -10 to C_VVC_CMD_DATA_MAX_LENGTH;
  -- Common VVC fields
  operation                 : t_operation;
  proc_call                 : string(1 to C_VVC_CMD_STRING_MAX_LENGTH);
  msg                       : string(1 to C_VVC_CMD_STRING_MAX_LENGTH);
  data_routing              : t_data_routing;
  cmd_idx                   : natural;
  command_type              : t_immediate_or_queued;
  msg_id                    : t_msg_id;
  gen_integer_array         : t_integer_array(0 to 1); -- Increase array length if needed
  gen_boolean               : boolean; -- Generic boolean
  timeout                   : time;
  alert_level               : t_alert_level;
  delay                     : time;
  quietness                 : t_quietness;
  parent_msg_id_panel       : t_msg_id_panel;
end record;
```

vvc_cmd_pkg / C_VVC_CMD_DEFAULT / Reset values



```
constant C_VVC_CMD_DEFAULT : t_vvc_cmd_record := (
  -- Default/reset values for VVC common fields
  data_array                => (others => (others => '0')),
  data_array_length         => 1,
  -- Common VVC fields
  operation                 => NO_OPERATION,
  proc_call                 => (others => NUL),
  msg                       => (others => NUL),
  data_routing              => NA,
  cmd_idx                   => 0,
  command_type              => NO_COMMAND_TYPE,
  msg_id                    => NO_ID,
  gen_integer_array         => (others => -1),
  gen_boolean               => false,
  timeout                   => 0 ns,
  alert_level               => FAILURE,
  delay                     => 0 ns,
  quietness                 => NON_QUIET,
  parent_msg_id_panel       => C_UNUSED_MSG_ID_PANEL
);
```

I2c_user_vvc / instance



```vhdl
entity i2c_user_vvc is
  generic (
    GC_INSTANCE_IDX                          : natural                   := 1;
    GC_I2C_USER_BFM_CONFIG                   : t_i2c_user_bfm_config     := C_I2C_USER_BFM_CONFIG_DEFAULT;
    GC_CMD_QUEUE_COUNT_MAX                   : natural                   := C_CMD_QUEUE_COUNT_MAX;
    GC_CMD_QUEUE_COUNT_THRESHOLD             : natural                   := C_CMD_QUEUE_COUNT_THRESHOLD;
    GC_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY    : t_alert_level             := C_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY;
    GC_RESULT_QUEUE_COUNT_MAX                : natural                   := C_RESULT_QUEUE_COUNT_MAX;
    GC_RESULT_QUEUE_COUNT_THRESHOLD          : natural                   := C_RESULT_QUEUE_COUNT_THRESHOLD;
    GC_RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY : t_alert_level             := C_RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY
  );
  port (
    clk                         : in    std_logic;
    i2c_user_vvc_if             : inout t_i2c_user_if := init_i2c_user_if_signals
  );
end entity i2c_user_vvc;
```

t_i2c_user_if (interface) type is in **i2c_user_bfm_pkg**

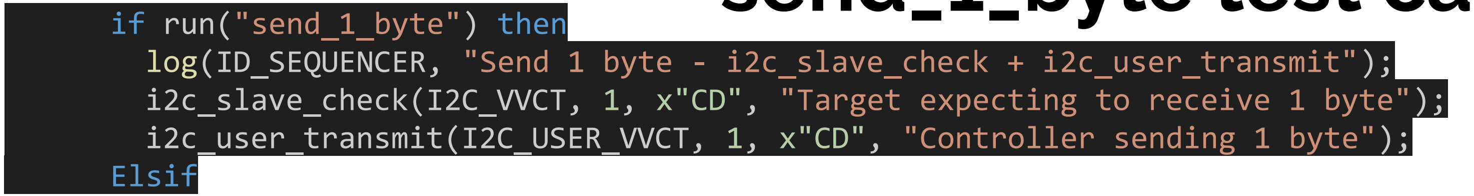
init_i2c_user_if_signals function is in **vvc_methods_pkg**

I2c_user_vvc / command operations

```
        when TRANSMIT =>
          i2c_user_transmit(
            msg         => format_msg(v_cmd),
            data_array  => v_cmd.data_array(0 to v_cmd.data_array_length -1),
            clk         => clk,
            i2c_user_if => i2c_user_vvc_if,
            scope       => C_SCOPE,
            config      => vvc_config.bfm_config
          );
```

```
        when RECEIVE =>
          i2c_user_receive(
            msg         => format_msg(v_cmd),
            data_array  => v_cmd.data_array(0 to v_cmd.data_array_length -1),
            clk         => clk,
            i2c_user_if => i2c_user_vvc_if,
            scope       => C_SCOPE,
            config      => vvc_config.bfm_config
          );
```

send_1_byte test case



```
      if run("send_1_byte") then
        log(ID_SEQUENCER, "Send 1 byte - i2c_slave_check + i2c_user_transmit");
        i2c_slave_check(I2C_VVCT, 1, x"CD", "Target expecting to receive 1 byte");
        i2c_user_transmit(I2C_USER_VVCT, 1, x"CD", "Controller sending 1 byte");
      Elsif
```

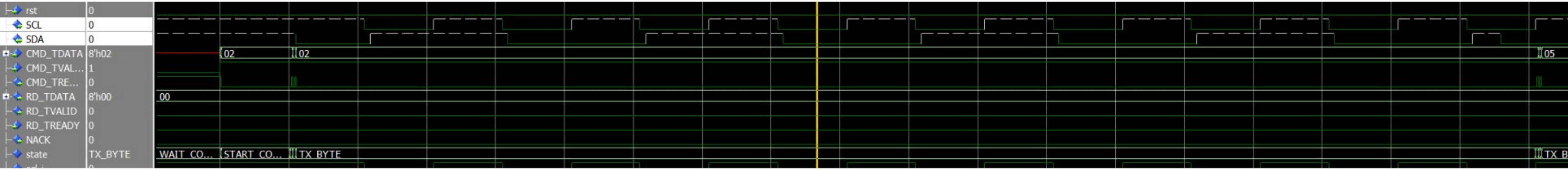
Questa transcript

```
# UVVM: ID_BFM       250000.0 ns  I2C_USER_VVC,1   i2c_user_transmit(x"CD") target_addr: 55 'Controller
# UVVM:                                              sending 1 byte'  [2]
# UVVM: ID_BFM       250000.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0x01
# UVVM: ID_BFM       250100.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM       255300.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0xAA
# UVVM: ID_BFM       255500.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM       345700.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0xCD
# UVVM: ID_BFM       345900.0 ns  I2C_USER_VVC,1   i2c_user_transmit(): receive cmd byte: 0x05
```

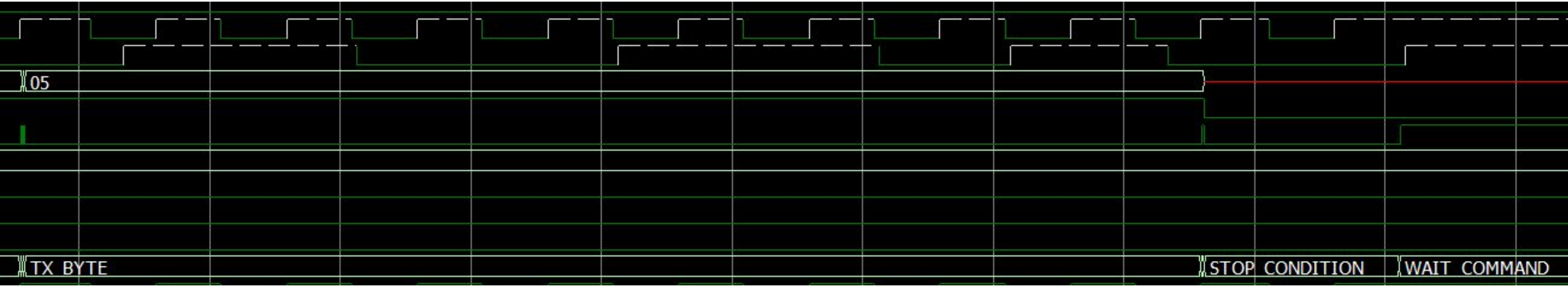
CMD_START_CONDITION +
CMD_TX_BYTE (TGTADR+W) +
CMD_TX_BYTE(CD) +
CMD_STOP_CONDITION

send_1_byte test case

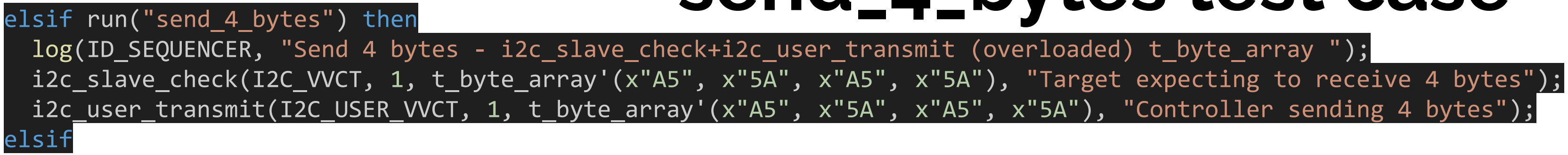
I2C_Address + WR - [10101010]



x”CD”[11001101]



send_4_bytes test case



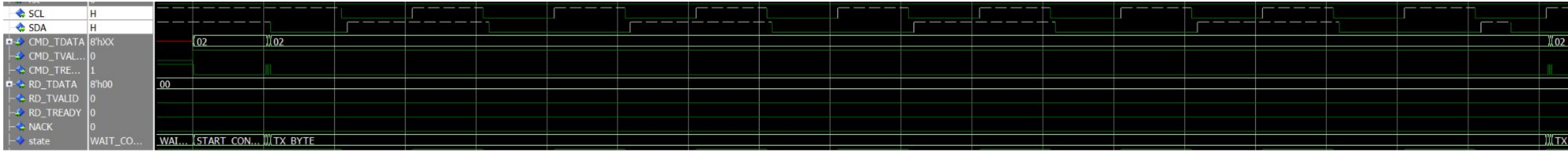
```
elsif run("send_4_bytes") then
  log(ID_SEQUENCER, "Send 4 bytes - i2c_slave_check+i2c_user_transmit (overloaded) t_byte_array ");
  i2c_slave_check(I2C_VVCT, 1, t_byte_array'(x"A5", x"5A", x"A5", x"5A"), "Target expecting to receive 4 bytes");
  i2c_user_transmit(I2C_USER_VVCT, 1, t_byte_array'(x"A5", x"5A", x"A5", x"5A"), "Controller sending 4 bytes");
elsif
```

Questa transcript

```
# UVVM: ID_BFM                 250000.0 ns  I2C_USER_VVC,1     i2c_user_transmit(x"A5", x"5A", x"A5", x"5A")
# UVVM:                                     target_addr: 55 'Controller sending 4 bytes'  [2]
# UVVM: ID_BFM                 250000.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x01
# UVVM: ID_BFM                 250100.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                 255300.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0xAA
# UVVM: ID_BFM                 255500.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x02
...
# UVVM: ID_BFM                 526700.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                 616900.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x5A
# UVVM: ID_BFM                 617100.0 ns  I2C_USER_VVC,1     i2c_user_transmit(): receive cmd byte: 0x05
# UVVM: ID_CMD_EXECUTOR_WAIT          707300.0 ns  I2C_USER_VVC,1      ..Executor: Waiting for command
# UVVM: ID_BFM                 722700.0 ns  I2C_VVC,1        i2c_slave_check((x"A5", x"5A", x"A5", x"5A"))=> OK, read
# UVVM:                                     data = (x"A5", x"5A", x"A5", x"5A"). 'Target expecting to
# UVVM:                                     receive 4 bytes'  [1]
```

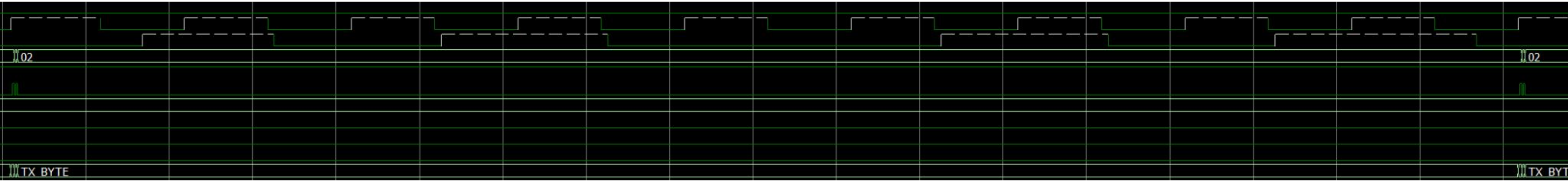
send_4_bytes test case

I2C_Address + WR - [10101010]

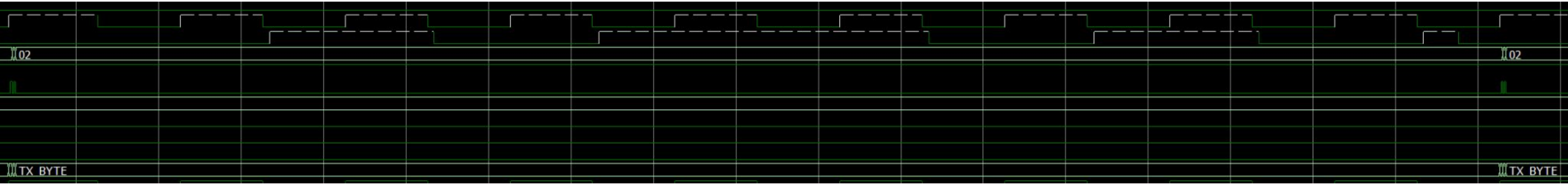


SCL	H
SDA	H
CMD_TDATA	8'hXX
CMD_TVAL...	0
CMD_TRE...	1
RD_TDATA	8'h00
RD_TVALID	0
RD_TREADY	0
NACK	0
state	WAIT_CO...

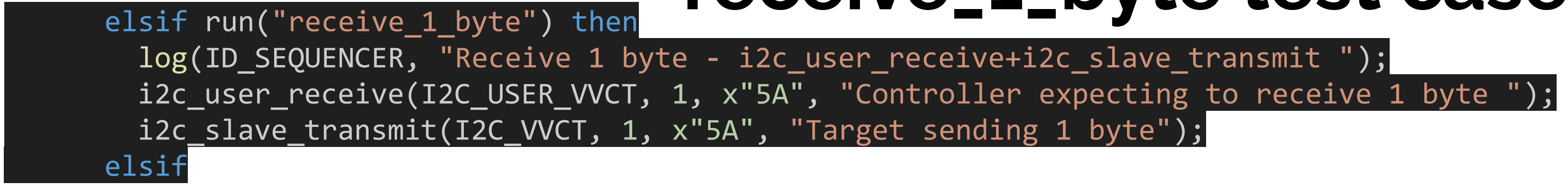
x”A5”[10100101]



x”5A”[01011010]



receive_1_byte test case



```
      elsif run("receive_1_byte") then
        log(ID_SEQUENCER, "Receive 1 byte - i2c_user_receive+i2c_slave_transmit ");
        i2c_user_receive(I2C_USER_VVCT, 1, x"5A", "Controller expecting to receive 1 byte ");
        i2c_slave_transmit(I2C_VVCT, 1, x"5A", "Target sending 1 byte");
      elsif
```

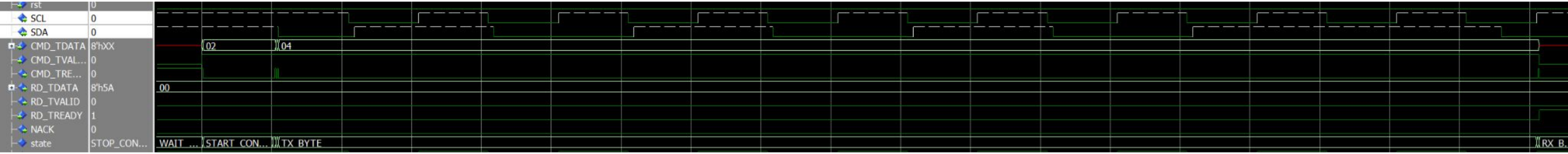
Questa transcript

```
# UVVM: ID_BFM           250000.0 ns  I2C_USER_VVC,1     i2c_user_receive(x"5A") target_addr: 55 'Controller
# UVVM:                              expecting to receive 1 byte '  [1]
# UVVM: ID_BFM           250100.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0x02
# UVVM: ID_BFM           255300.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0xAB
# UVVM: ID_BFM           255500.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0x04

# UVVM: ID_POS_ACK          435800.0 ns  TB seq.         check_value() => OK, for slv x"5A"'. 'Received data
# UVVM:                              should match expected'
```

CMD_START_CONDITION +
CMD_RX_BYTE (TGTADR+R) +
CMD_RX_BYTE_NACK

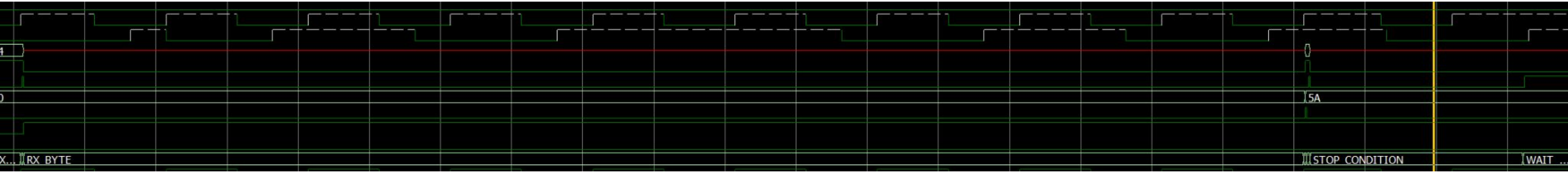
receive_1_byte test case

I2C_Address + RD - [10101011]

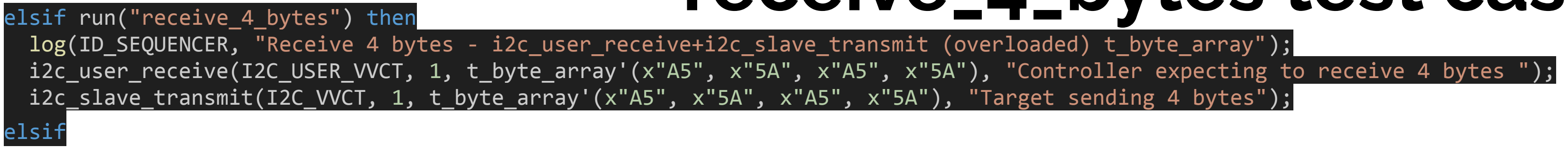


rst	0
SCL	0
SDA	0
CMD_TDATA	8'hXX
CMD_TVAL...	0
CMD_TRE...	0
RD_TDATA	8'h5A
RD_TVALID	0
RD_TREADY	1
NACK	0
state	STOP_CON...

x”5A”[01011010]



receive_4_bytes test case



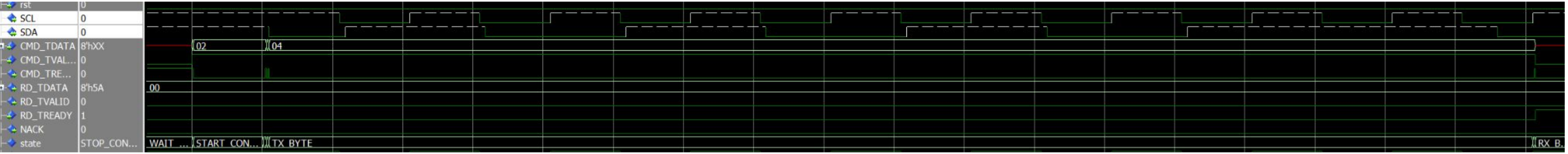
```
elsif run("receive_4_bytes") then
  log(ID_SEQUENCER, "Receive 4 bytes - i2c_user_receive+i2c_slave_transmit (overloaded) t_byte_array");
  i2c_user_receive(I2C_USER_VVCT, 1, t_byte_array'(x"A5", x"5A", x"A5", x"5A"), "Controller expecting to receive 4 bytes ");
  i2c_slave_transmit(I2C_VVCT, 1, t_byte_array'(x"A5", x"5A", x"A5", x"5A"), "Target sending 4 bytes");
elsif
```

Questa transcript

```
# UVVM: ID_BFM          250000.0 ns  I2C_USER_VVC,1     i2c_user_receive(x"A5", x"5A", x"A5", x"5A") target_addr:
# UVVM:                                55 'Controller expecting to receive 4 bytes '  [1]
# UVVM: ID_BFM          250100.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0x02
# UVVM: ID_BFM          255300.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0xAB
# UVVM: ID_BFM          255500.0 ns  I2C_USER_VVC,1     i2c_user_receive(): receive cmd byte: 0x03
# UVVM: ID_POS_ACK        435800.0 ns  TB seq.          check_value() => OK, for slv x"A5"'. 'Received data
# UVVM:                                should match expected'
# UVVM: ID_POS_ACK        526200.0 ns  TB seq.          check_value() => OK, for slv x"5A"'. 'Received data
# UVVM:                                should match expected'
# UVVM: ID_POS_ACK        616600.0 ns  TB seq.          check_value() => OK, for slv x"A5"'. 'Received data
# UVVM:                                should match expected'
# UVVM: ID_POS_ACK        707000.0 ns  TB seq.          check_value() => OK, for slv x"5A"'. 'Received data
# UVVM:                                should match expected'
```

send_4_bytes test case

I2C_Address + RD - [10101011]

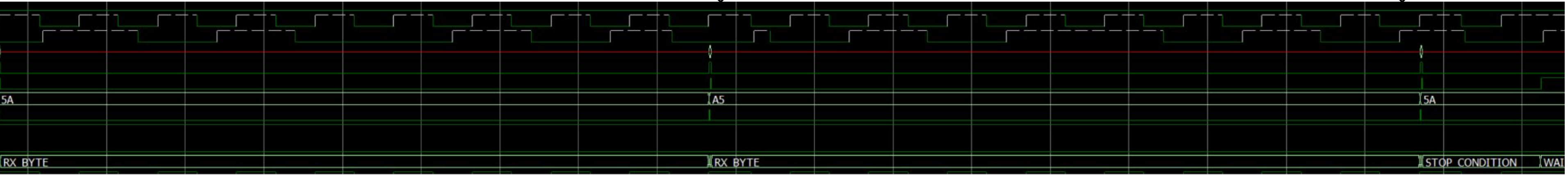


rst 0
SCL 0
SDA 0
CMD_TDATA 8'hXX
CMD_TVAL... 0
CMD_TRE... 0
RD_TDATA 8'h5A
RD_TVALID 0
RD_TREADY 1
NACK 0
state STOP_CON...

x”5A”[01011010] - last byte NACK

ACK

NACK



OSVVM Setup

Declare the library in testbench

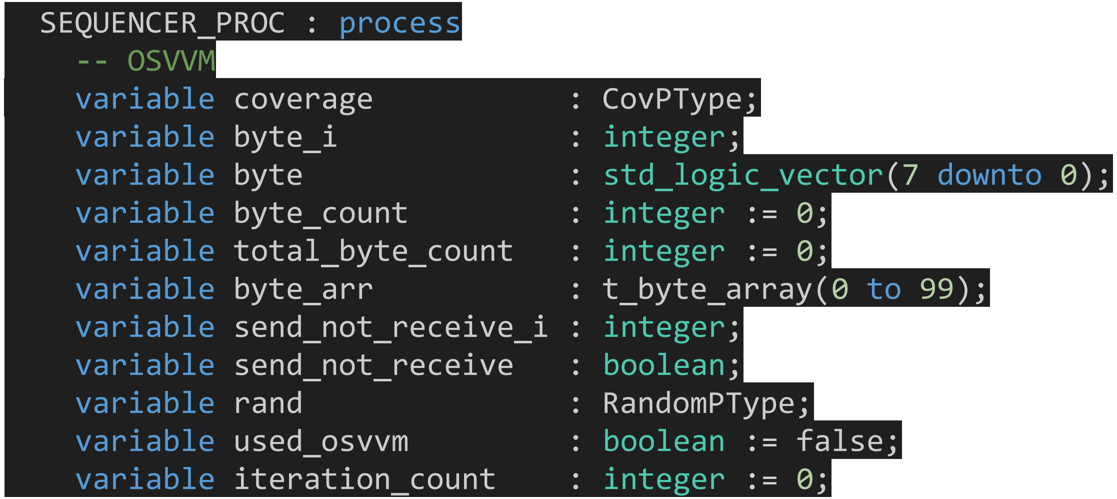
```
-- OSVVM
library osvvm;
use osvvm.CoveragePkg.all;
use osvvm.AlertLogPkg.all;
use osvvm.RandomPkg.all;
```

Setup OSVVM by setting the AlertStopCount to 1 error and start the random seed generation

```
    --------------------
    -- OSVVM setup
    --------------------
    SetAlertStopCount(ERROR,1);
    rand.InitSeed(rand'instance_name);
    --------------------
```

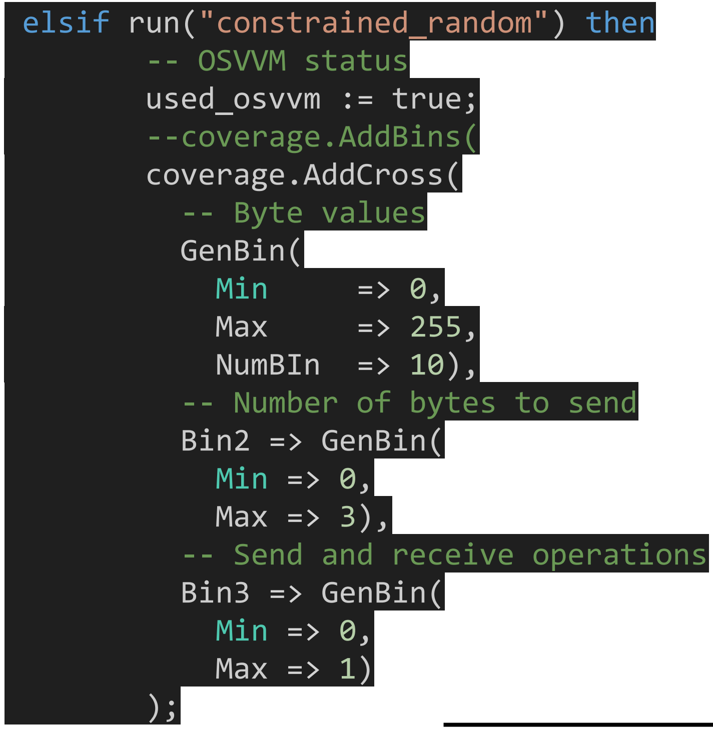
OSVVM variable(s) defiition

Define local variables at the sequencer



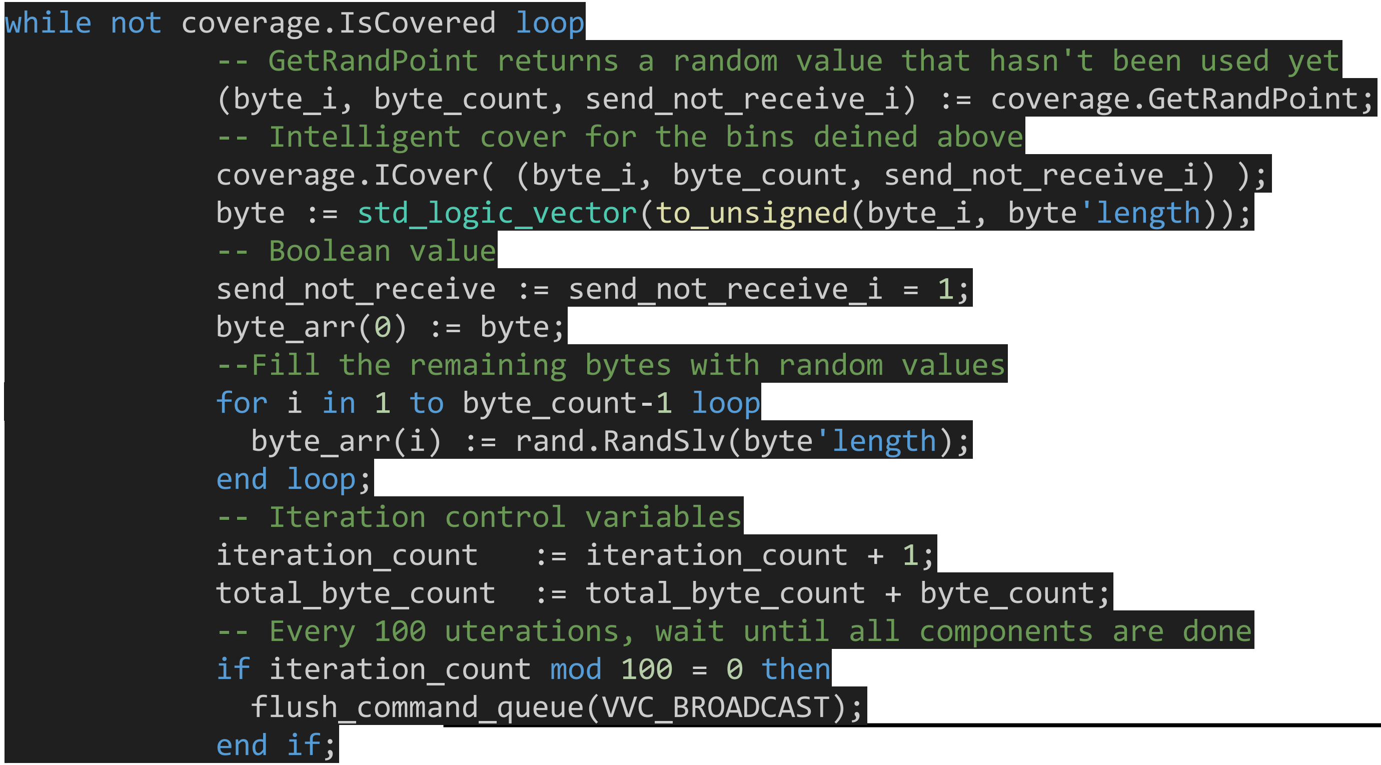
```
  SEQUENCER_PROC : process
    -- OSVVM
    variable coverage           : CovPType;
    variable byte_i             : integer;
    variable byte               : std_logic_vector(7 downto 0);
    variable byte_count         : integer := 0;
    variable total_byte_count   : integer := 0;
    variable byte_arr           : t_byte_array(0 to 99);
    variable send_not_receive_i : integer;
    variable send_not_receive   : boolean;
    variable rand               : RandomPType;
    variable used_osvvm         : boolean := false;
    variable iteration_count    : integer := 0;
```

constrain_random test case



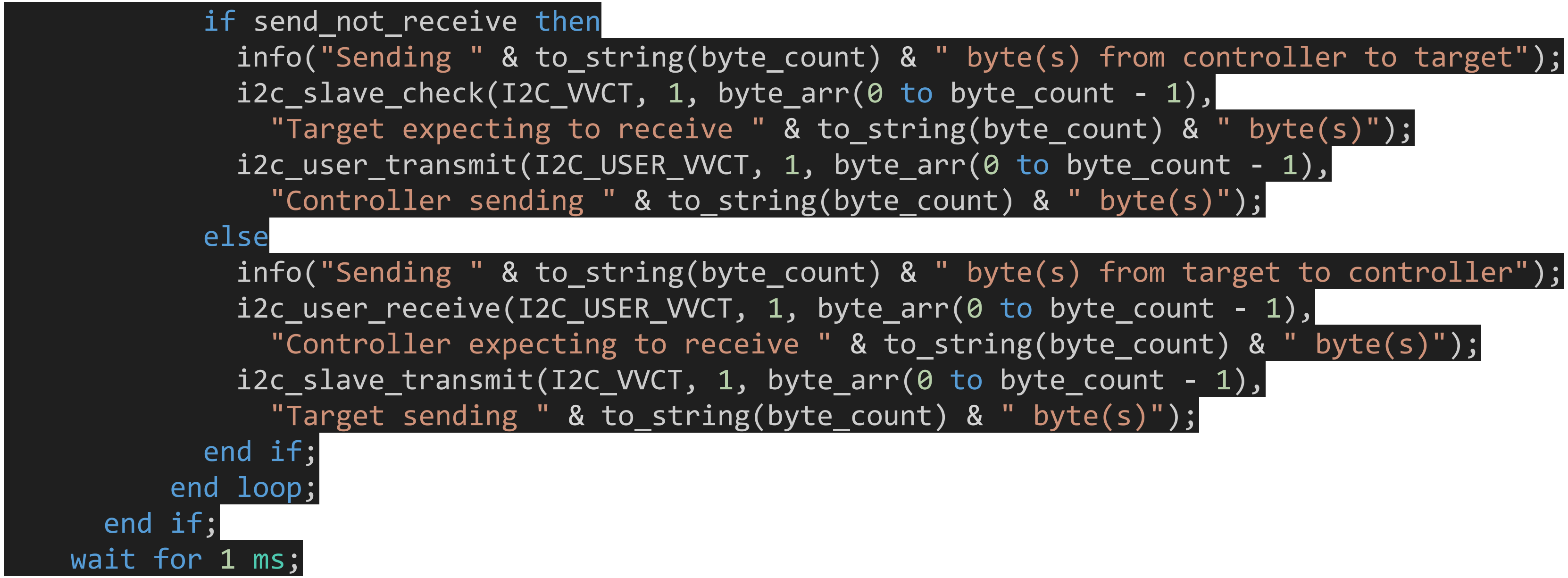
```
elsif run("constrained_random") then
       -- OSVVM status
       used_osvvm := true;
       --coverage.AddBins(
       coverage.AddCross(
         -- Byte values
         GenBin(
           Min     => 0,
           Max     => 255,
           NumBIn  => 10),
         -- Number of bytes to send
         Bin2 => GenBin(
           Min => 0,
           Max => 3),
         -- Send and receive operations
         Bin3 => GenBin(
           Min => 0,
           Max => 1)
       );
```

constrain_random test case



```
while not coverage.IsCovered loop
            -- GetRandPoint returns a random value that hasn't been used yet
            (byte_i, byte_count, send_not_receive_i) := coverage.GetRandPoint;
            -- Intelligent cover for the bins deined above
            coverage.ICover( (byte_i, byte_count, send_not_receive_i) );
            byte := std_logic_vector(to_unsigned(byte_i, byte'length));
            -- Boolean value
            send_not_receive := send_not_receive_i = 1;
            byte_arr(0) := byte;
            --Fill the remaining bytes with random values
            for i in 1 to byte_count-1 loop
              byte_arr(i) := rand.RandSlv(byte'length);
            end loop;
            -- Iteration control variables
            iteration_count   := iteration_count + 1;
            total_byte_count  := total_byte_count + byte_count;
            -- Every 100 uterations, wait until all components are done
            if iteration_count mod 100 = 0 then
              flush_command_queue(VVC_BROADCAST);
            end if;
```

constrain_random test case



```
            if send_not_receive then
              info("Sending " & to_string(byte_count) & " byte(s) from controller to target");
              i2c_slave_check(I2C_VVCT, 1, byte_arr(0 to byte_count - 1),
                "Target expecting to receive " & to_string(byte_count) & " byte(s)");
              i2c_user_transmit(I2C_USER_VVCT, 1, byte_arr(0 to byte_count - 1),
                "Controller sending " & to_string(byte_count) & " byte(s)");
            else
              info("Sending " & to_string(byte_count) & " byte(s) from target to controller");
              i2c_user_receive(I2C_USER_VVCT, 1, byte_arr(0 to byte_count - 1),
                "Controller expecting to receive " & to_string(byte_count) & " byte(s)");
              i2c_slave_transmit(I2C_VVCT, 1, byte_arr(0 to byte_count - 1),
                "Target sending " & to_string(byte_count) & " byte(s)");
            end if;
          end loop;
      end if;
    wait for 1 ms;
```

OSVVM cleanup

```
    --------------------
    -- OSVVM cleanup
    --------------------
    if used_osvvm then
      info("OSVVM - All coverage points met");
      info("Iterations:               " & to_string(iteration_count));
      info("Send and received bytes:  " & to_string(total_byte_count));
      info("Errors and warnings       " & to_string(GetAlertCount));
    end if;
```

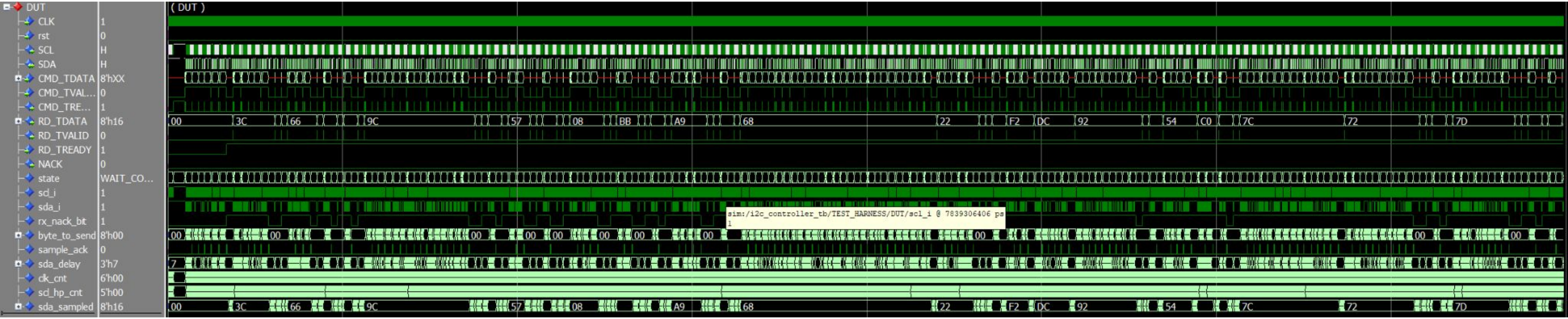
Questa transcript

```
...
#    250000000 ps - default          -   INFO - Sending 1 byte(s) from target to controller
# UVVM: ID_CMD_INTERPRETER_WAIT          250000.0 ns  I2C_VVC,1            ..Interpreter: Waiting for command
# UVVM: ID_UVVM_SEND_CMD              250000.0 ns  TB seq.(uvvm)       ->i2c_user_receive(I2C_USER_VVC,1,    1 bytes:
# UVVM:                                          'Controller expecting to receive 1 byte(s)'. [159]
# UVVM: ID_CMD_INTERPRETER             250000.0 ns  I2C_USER_VVC,1        i2c_user_receive(I2C_USER_VVC,1,    1 bytes. Command
# UVVM:                                          received  [159]
# UVVM: ID_UVVM_CMD_ACK               250000.0 ns  TB seq.(uvvm)          ACK received.   [159]
# UVVM: ID_CMD_INTERPRETER_WAIT          250000.0 ns  I2C_USER_VVC,1        ..Interpreter: Waiting for command
# UVVM: ID_UVVM_SEND_CMD              250000.0 ns  TB seq.(uvvm)       ->i2c_slave_transmit(I2C_VVC,1): 'Target sending 1
# UVVM:                                          byte(s)'. [160]
# UVVM: ID_CMD_INTERPRETER             250000.0 ns  I2C_VVC,1           i2c_slave_transmit(I2C_VVC,1). Command received  [160]
# UVVM: ID_UVVM_CMD_ACK               250000.0 ns  TB seq.(uvvm)          ACK received.   [160]
# UVVM: ID_CMD_INTERPRETER_WAIT          250000.0 ns  I2C_VVC,1            ..Interpreter: Waiting for command
# UVVM: ID_BFM                   250100.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                   255300.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0xAA
# UVVM: ID_BFM                   255500.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                   345700.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0xEF
# UVVM: ID_BFM                   345900.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                   436100.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0xF4
# UVVM: ID_BFM                   436300.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0x02
# UVVM: ID_BFM                   526500.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0xF7
# UVVM: ID_BFM                   526700.0 ns  I2C_USER_VVC,1      i2c_user_transmit(): receive cmd byte: 0x05
```

Questa transcript

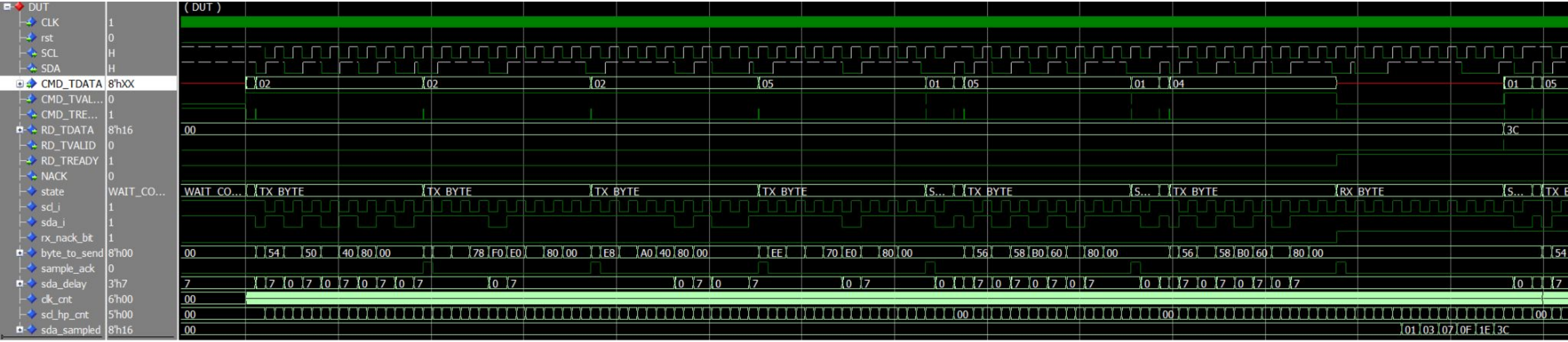
```
====================================================================================================
# UVVM:     *** FINAL SUMMARY OF ALL ALERTS ***
# UVVM:
====================================================================================================
# UVVM:                          REGARDED   EXPECTED  IGNORED      Comment?
# UVVM:          NOTE         :     0       0       0       ok
# UVVM:          TB_NOTE      :     0       0       0       ok
# UVVM:          WARNING      :     0       0       0       ok
# UVVM:          TB_WARNING   :     0       0       0       ok
# UVVM:          MANUAL_CHECK :     0       0       0       ok
# UVVM:          ERROR        :     0       0       0       ok
# UVVM:          TB_ERROR     :     0       0       0       ok
# UVVM:          FAILURE      :     0       0       0       ok
# UVVM:          TB_FAILURE   :     0       0       0       ok
# UVVM:
====================================================================================================
# UVVM:     >> Simulation SUCCESS: No mismatch between counted and expected serious alerts
# UVVM:
====================================================================================================
# UVVM:
# UVVM:
#   19962300000 ps - default          -   INFO - OSVVM - All coverage points met
#   19962300000 ps - default          -   INFO - Iterations:            80
#   19962300000 ps - default          -   INFO - Send and received bytes:  120
#   19962300000 ps - default          -   INFO - Errors and warnings      0
```

Wave



```
DUT
  CLK           1
  rst           0
  SCL           H
  SDA           H
  CMD_TDATA     8'hXX
  CMD_TVAL...   0
  CMD_TRE...    1
  RD_TDATA      8'h16
  RD_TVALID     0
  RD_TREADY     1
  NACK          0
  state         WAIT_CO...
  scl_i         1
  sda_i         1
  rx_nack_bit   1
  byte_to_send  8'h00
  sample_ack    0
  sda_delay     3'h7
  clk_cnt       6'h00
  scl_hp_cnt    5'h00
  sda_sampled   8'h16
```

```
sim:/i2c_controller_tb/TEST_HARNESS/DUT/scl_i @ 7839306406 ps
1
```



```
DUT
  CLK           1
  rst           0
  SCL           H
  SDA           H
  CMD_TDATA     8'hXX
  CMD_TVAL...   0
  CMD_TRE...    1
  RD_TDATA      8'h16
  RD_TVALID     0
  RD_TREADY     1
  NACK          0
  state         WAIT_CO...
  scl_i         1
  sda_i         1
  rx_nack_bit   1
  byte_to_send  8'h00
  sample_ack    0
  sda_delay     3'h7
  clk_cnt       6'h00
  scl_hp_cnt    5'h00
  sda_sampled   8'h16
```

Advanced VHDL Verification by combining testing frameworks

VUnit + UVVM + OSVVM