UVVM Essential Mechanisms – Quick Reference

This document explains some of the essential mechanisms necessary for running UVVM, in addition to helpful and important VVC status and configuration records which are accessible directly from the testbench. More details on the VVC Framework and the command mechanism can be found in the VVC Framework Manual.

VHDL 2008 only

1 Libraries

In order to use a VVC the following libraries need to be included:

```
library uvvm_util;
context uvvm_util.uvvm_util_context;

library uvvm_vvc_framework;
use uvvm_vvc_framework.ti_vvc_framework_support_pkg.all;

library bitvis_vip_<name>;
context bitvis_vip_<name>.vvc_context;
```

2 UVVM Initialization

The following mechanisms are required for running UVVM

Mechanism	Description
ti_uvvm_engine	**ti_uvvm_engine** This entity contains a process that will initialize the UVVM environment, and has to be instantiated in the testbench harness, or alternatively in the top-level testbench. Example: `i_ti_uvvm_engine : entity uvvm_vvc_framework.ti_uvvm_engine;`
await_uvvm_initialization()	**await_uvvm_initialization(VOID)** This procedure is a blocking procedure that has to be called from the testbench sequencer, prior to any VVC calls, to ensure that the UVVM engine has been initialized and is ready. This procedure will check the shared_uvvm_state on each delta cycle until the UVVM engine has been initialized. Note that this method is depending on the ti_uvvm_engine mechanism. Note that this method uses the `t_void` parameter, defined in the UVVM Utility Library types package. Example: `await_uvvm_initialization(VOID);`

3 UVVM and VVC Shared Variables

UVVM and VVC shared variables are defined in global_signals_and_shared_variables_pkg and the various vvc_methods_pkg, respectively.

Shared variables

Shared variable	Description

shared_uvvm_status	Shared variable providing access to VVC related information via the `info_on_finishing_await_any_completion` record element, i.e.
	`shared_uvvm_status.info_on_finishing_await_any_completion`
	
	This record element gives access to the name, command index and the time of completion of the VVC that first fulfilled the `await_any_completion()`. The available record fields are:
	
	`vvc_name : string -- default "no await_any_completion() yet"`
	`vvc_cmd_idx : natural -- default 0`
	`vvc_time_of_completion : time -- default 0 ns`
	
	For more information regarding other fields available in the `shared_uvvm_status` see the UVVM Util QuickRef, section 1.4

shared_<vvc_name>_vvc_config	Shared variable providing access to configuration parameters for each VVC instance and channel if applicable. E.g.
	
	`shared_sbi_vvc_config(1).inter_bfm_delay.delay_type := TIME_START2START;`
	`shared_uart_vvc_config(RX,1).bfm_config.bit_time := C_BIT_TIME;`

shared_<vvc_name>_vvc_status	Shared variable providing access to status parameters for each VVC instance and channel if applicable. E.g.
	
	`v_num_pending_cmds := shared_sbi_vvc_status(1).pending_cmd_cnt;`
	`v_current_cmd_idx := shared_uart_vvc_status(TX,2).current_cmd_idx;`
	`v_previous_cmd_idx := shared_uart_vvc_status(TX,2).previous_cmd_idx;`

shared_<vvc_name>_transaction_info	Shared variable providing access to VVC instances transaction information to include in the wave view during simulation.
	Available information is dependent on VVC type and typical information is:
	
	`operation : t_operation; -- default NO_OPERATION`
	`data : std_logic_vector(C_VVC_CMD_DATA_MAX_LENGTH-1 downto 0); -- default 0x0`
	`msg : string(1 to C_VVC_CMD_STRING_MAX_LENGTH); -- default empty`

4 VVC Status, Configuration and Transaction information

The VVC status, configuration and transaction information records are defined in each individual VVC methods package.

Each VVC instance and channel can be configured and useful information can be accessed from the testbench via dedicated shared variables.

From the VVC configuration shared variable, one is given the ability to tailor each VVC to one's needs, in addition to access the BFM configuration record via the `bfm_config` identifier. In addition to BFM configuration possibility, the configuration settings consist of command and result queue settings, BFM access separation delay and a VVC dedicated message ID panel.
Note that some BFMs require user configuration, e.g. the `bit_time` setting in serial interface BFMs.

The VVC status shared variable provide access to the command status parameters for each of the VVCs, such as the current and previous command index, and the number of pending commands in the VVCs command queue. This provide a helpful tool, e.g. when synchronizing VVCs in the test sequencer using the `await_completion()` or `await_any_completion()` methods.

When using a wave viewer during simulation, the transaction shared variable provides helpful information regarding current VVC operation and transaction information such as `address` and `data`. Note that the accessible fields depend on the VVC and its implementation. An example of two SBI VVCs performing FIFO write operations, followed by check operations, is shown in Figure 4-1.

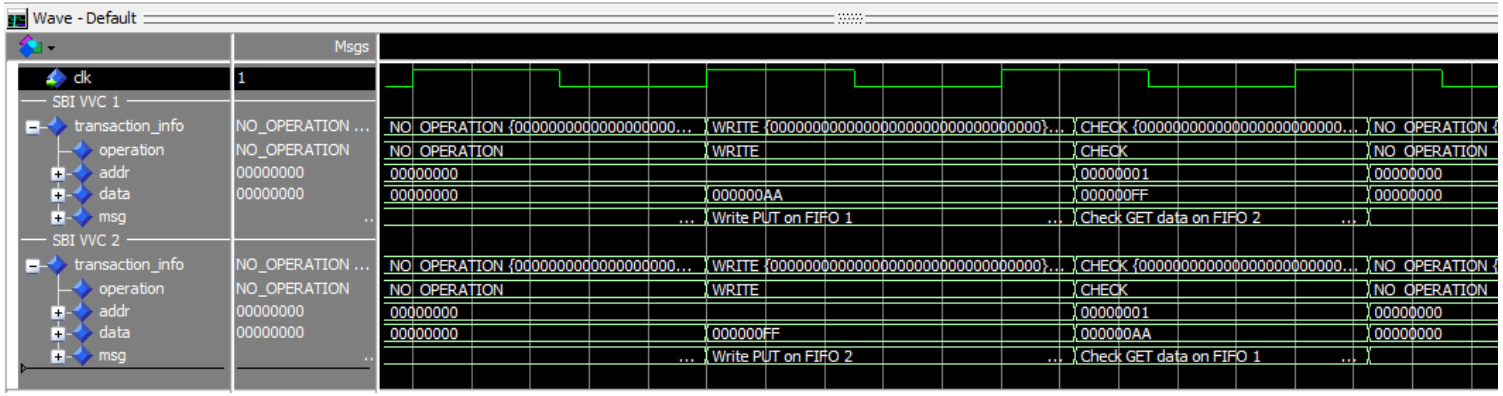

Figure 4-1 VVC Transaction info example

5 Activity Watchdog

UVVM VVC Framework has an activity watchdog mechanism which all UVVM VVCs support. All VVCs can be automatically registered in the activity watchdog register at start-up and will during simulations update the activity watchdog with their current activity status, i.e. active or inactive. A timeout counter will start when no VVC activity is registered in the activity watchdog and is reset with VVC activity. An alert will be raised if no VVC has an activity prior to the timeout counter reaching the specified timeout value.

The activity watchdog has to be included as a concurrent procedure parallel to the testbench sequencer or in the test harness in order to be activated.
Note that the activity watchdog will raise a TB_WARNING if the number of expected VVCs does not match the number of registered VVCs in the activity watchdog register, and that leaf VVCs (e.g. channels) are counted individually.

Note that some VVC activity is ignored by the activity watchdog. This currently applies to the clock generator VVC, as this VVC may continue to be active even after all other testbench activity has stopped. This VVC will have to be included in the number of expected VVCs registered in the activity watchdog register, but will not have any effect on the activity watchdog timeout counter.

Field name	Type name	Default	Description
num_exp_vvc	natural		Expected number of VVCs which should be registered in activity watchdog VVC register (including any clock generator VVC). Note that each channel is counted in the number of registered VVCs in the activity watchdog register.
timeout	time		Timeout value after last VVC activity.
alert_level	t_alert_level	TB_ERROR	The timeout will have this alert level.
msg	string	"AW_1"	Message included with activity watchdog log messages, e.g. name of activity watchdog

Example:

```
p_activity_watchdog:
    activity_watchdog(num_exp_vvc   => 3,
                      timeout       => C_ACTIVITY_WATCHDOG_TIMEOUT);
```

6 Compile scripts

In the script folder in the root directory the compile_all.do compiles all UVVM components. This script may be called with one to three input arguments. The first input argument is the directory of the script folder at the root directory from the working directory. The second input argument is the target directory of the compiled libraries, by default every library is compiled in a sim folder in the corresponding components directory. The third input argument is the directory to a custom component list in .txt format. The script will only compile the components listed in that file. By default, the script uses the file component_list.txt located in uvvm/script. This file can be modified so that only some components are compiled.

Example: do uvvm/script/compile_all.do uvvm/script

There are also compile scripts for all UVVM components located in the script folder of each UVVM component. These scripts can be called with two input arguments. The first input argument is the directory of the component folder from the working directory. The second input argument is the target directory of the compiled library, default is the sim folder in the respective component.

Example: do uvvm/uvvm_util/script/compile_src.do uvvm/uvvm_util

7 Scope of verbosity control

Message IDs are used for verbosity control in many of the procedures and functions in UVVM, as well as log messages and checks in VVCs, BFMs and Scoreboards.
Note that VVCs and Scoreboards comes with dedicated message ID panels and are not affected by the global message ID panel, but accessed by addressing targeting VVC or Scoreboard and, if applicable, instance number or with a broadcast.

E.g.

```
-- Global message ID panel. Does not apply to VVCs or Scoreboards, as they have their own local message ID panel
disable_log_msg(ALL_MESSAGES);
enable_log_msg(ID_SEQUENCER);

-- VVC message ID panel
disable_log_msg(VVC_BROADCAST, ALL_MESSAGES); -- broadcast to all VVCs and instances
enable_log_msg(I2C_VVCT, C_VVC_INSTANCE_1, ID_BFM_WAIT); -- I2C VVC instance 1
enable_log_msg(I2C_VVTC, C_VVC_INSTANCE_2, ID_BFM_WAIT); -- I2C VVC instance 2

-- Scoreboard message ID panel
shared variable sb_under_test : record_sb_pkg.t_generic_sb;
…
sb_under_test.disable_log_msg(ALL_INSTANCES, ID_CTRL); -- broadcast to all SB instances
sb_under_test.enable_log_msg(C_SB_INSTANCE_1, ID_DATA); -- SB instance 1
```

A subset of message IDs is listed in UVVM Utility Library QR, section 1.10.