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. A more comprehensive review can be found in the VVC Framework Manual.

VHDL 2008 only

1 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);`

2 UVVM and VVC Shared Variables

UVVM and VVC shared variables are defined in the UVVM methods package and VVC methods package, 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`

3 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 3-1.

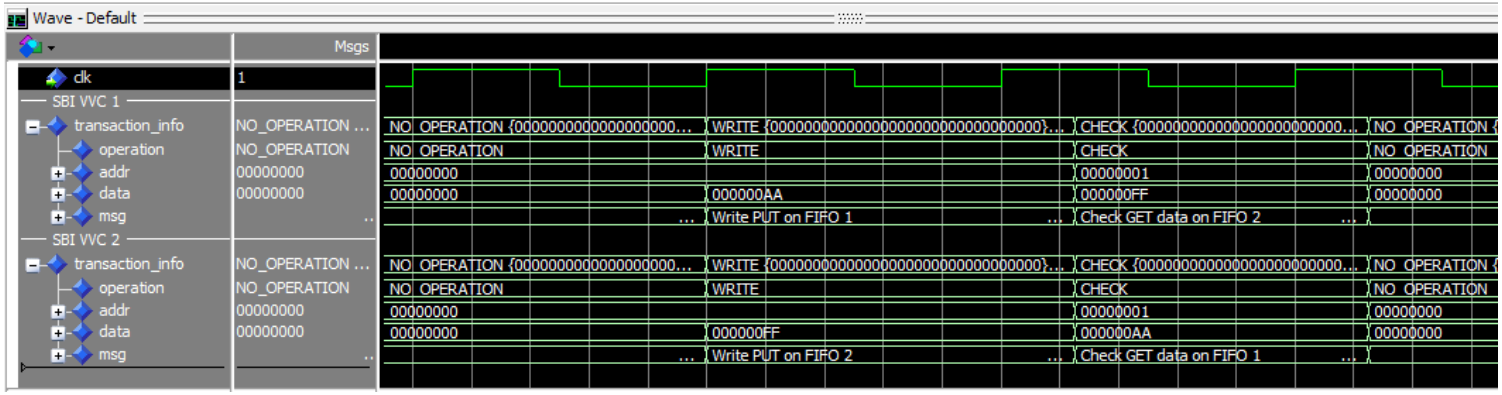

Figure 3-1 VVC Transaction info example

4 Multiple Central Sequencers

A structured test environment is important and we recommend the use of a test harness to instantiate VVCs, DUT, clock generator and so forth. The testbench may consist of one or more test sequencers which are used to control the complete testbench architecture with any number of VVCs, although for a better testbench overview we recommend to have a single central test sequencer only.