**Ethernet VVC** –Quick Reference

**VVC**

This ethernet VVC is based on IEEE 802.3. It does not support optional fields or EtherType, only length is supported.

For general information see UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc.

|  |
| --- |
| ethernet\_send (VVCT, vvc\_instance\_idx, channel, [mac\_dest], [mac\_src], payload, msg) |
| Example: ethernet\_send(ETHERNET\_VVCT, 1, SEND, v\_mac\_dest, v\_mac\_src, v\_payload, “Send ethernet packet”); |

*ethernet\_vvc.vhd*

|  |
| --- |
| ethernet\_receive (VVCT, vvc\_instance\_idx, channel, msg) |
| Example: ethernet\_receive(ETHERNET\_VVCT, 1, RECEIVE, “Receive ethernet packet”); |

|  |
| --- |
| ethernet\_expect (VVCT, vvc\_instance\_idx, channel, [mac\_dest], [mac\_src], payload, msg, [alert\_level]) |
| Example: ethernet\_expect(ETHERNET\_VVCT, 1, RECEIVE, v\_mac\_dest, v\_mac\_src, v\_payload, “Expect ethernet packet”, ERROR); |

Ethernet VVC Configuration record **´vvc\_config´ --** accessible via **shared\_ethernet\_vvc\_config**

**Common VVC procedures applicable for this VVC**  
- See UVVM Methods QuickRef for details.

**await\_completion**() **enable\_log\_msg**() **disable\_log\_msg**()

**fetch\_result**()

**flush\_command\_queue**()  
**terminate\_current\_command**() **terminate\_all\_commands**() **insert\_delay**()

**get\_last\_received\_cmd\_idx()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record element** | **Type** | | **C\_SBI\_VVC\_CONFIG\_DEFAULT** | | |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | | C\_ETHERNET\_INTER\_BFM\_DELAY\_DEFAULT | | |
| cmd\_queue\_count\_max | natural | | C\_CMD\_QUEUE\_COUNT\_MAX | | |
| cmd\_queue\_count\_threshold | natural | | C\_CMD\_QUEUE\_COUNT\_THRESHOLD | | |
| cmd\_queue\_count\_threshold\_severity | t\_alert\_level | | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | | |
| result\_queue\_count\_max | natural | | C\_RESULT\_QUEUE\_COUNT\_MAX | | |
| result\_queue\_count\_threshold | natural | | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD | | |
| result\_queue\_count\_threshold\_severity | t\_alert\_level | | C\_RESULT\_QUEUE\_COUNT\_THERSHOLD\_SEVERITY | | |
| bfm\_config | t\_ethernet\_bfm\_config | | C\_ETHERNET\_BFM\_CONFIG\_DEFAULT | | |
| msg\_id\_panel | t\_msg\_id\_panel | | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | | |
|  |  | |  | | |
| BFM Configuration record ´**t\_ethernet\_bfm\_config´** | | | | | |
| **Record element** | | | **Type** | | **C\_ETHERNET\_BFM\_CONFIG\_DEFAULT** |
| mac\_destination | | | t\_byte\_array(0 to 5) | | x”000000000000” |
| mac\_source | | | t\_byte\_array(0 to 5) | | x”000000000000” |
| fcs\_error\_severity | | | t\_alert\_level | | TB\_ERROR |
| interpacket\_gap\_time | | | time | | 96 ns |

|  |  |
| --- | --- |
| Record ´**t\_ethernet\_frame´** | |
| **Record element** | **Type** |
| mac\_destination | t\_byte\_array(0 to 5) |
| mac\_source | t\_byte\_array(0 to 5) |
| length | t\_byte\_array(0 to 1) |
| payload | t\_byte\_array |
| fcs | t\_byte\_array(0 to 3) |

Ethernet VVC Status record signal **´vvc\_status´ --** accessible via **shared\_ethernet\_vvc\_status**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** |  |
| current\_cmd\_idx | natural |  | |
| previous\_cmd\_idx | natural |  |
| pending\_cmd\_cnt | natural |  |

VVC target parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| VVCT | t\_vvc\_target\_record | ETHERNET\_VVCT | VVC target type compiled into each VVC in order to differentiate between VVCs. |
| vvc\_instance\_idx | integer | 1 | Instance number of the VVC. |
| channel | t\_channel | SEND, RECEIVE | The VVC channel of the VVC instance. |

VVC functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| mac\_dest | t\_byte\_array |  | The MAC address of destination. |
| mac\_src | t\_byte\_array |  | The MAC address of source. |
| payload | t\_byte\_array |  | The payload containing data. |
| ethernet\_frame | t\_ethernet\_frame |  | The decoded ethernet frame. |

VVC transaction info

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| operation | t\_operation | Command operation type. |
| msg | string | Command message. |
| ethernet\_frame | t\_ethernet\_frame | Ethernet frame. |

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_INSTANCE\_IDX | natural | 1 | Instance number to assign the VVC |
| GC\_ETHERNET\_BFM\_CONFIG | t\_ethernet\_bfm\_config | C\_ETHERNET\_BFM\_CONFIG\_DEFAULT | Configuration for the Ethernet BFM. |
| GC\_CMD\_QUEUE\_COUNT\_MAX | natural | 1000 | Absolute maximum number of commands in the VVC command queue |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | An alert will be generated when reaching this threshold to indicate that the command queue is almost full. The queue will still accept new commands until it reaches C\_CMD\_QUEUE\_COUNT\_MAX. |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | Alert severity which will be used when command queue reaches GC\_CMD\_QUEUE\_COUNT\_THRESHOLD. |
| GC\_RESULT\_QUEUE\_COUNT\_MAX | natural | 1000 | Maximum number of unfetched results before result\_queue is full. |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | 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. |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | Severity of alert to be initiated if exceeding result\_queue\_count\_threshold |

VVC details

All VVC procedures are defined in vvc\_methods\_pkg (dedicated this VVC), and uvvm\_vvc\_framework.uvvm\_methods\_pkg and uvvm\_vvc\_framework.uvvm\_support\_pkg (common VVC procedures)

It is also possible to send a multicast to all instances of a VVC with ALL\_INSTANCES as parameter for vvc\_instance\_idx.

# VVC procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **ethernet\_send()** | **ethernet\_send (VVCT, vvc\_instance\_idx, channel, [mac\_dest], [mac\_src], payload, msg)**  The ethernet\_send() VVC procedure adds a send command to the Ethernet VVC executor queue, which will run as soon as all preceding commands have completed.  Example:  ethernet\_send(ETHERNET\_VVCT, 1, SEND, v\_mac\_dest, v\_mac\_src, v\_payload, “Send ethernet packet”);  ethernet\_send(ETHERNET\_VVCT, 1, SEND, v\_payload, “Send ethernet packet”); |
| **ethernet\_receive()** | **ethernet\_receive (VVCT, vvc\_instance\_idx, channel, msg)**  The ethernet\_receive() VVC procedure adds a receive command to the Ethernet VVC executor queue, which will run as soon as all preceding commands have completed.  The value read from DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the read data will be stored in the VVC for a potential future fetch (see example with *fetch\_result* below).  Example:  ethernet\_receive(ETHERNET\_VVCT, 1, RECEIVE, “Receive ethernet packet”);  **Example with fetch\_result() call**: Result is placed in **v\_data**  variable v\_cmd\_idx : natural; -- Command index for the last read  variable v\_data : bitvis\_vip\_ethernet.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read.  (…)  ethernet\_receive(ETHERNET\_VVCT, 1, RECEIVE, “Receive ethernet package”);  v\_cmd\_idx := shared\_cmd\_idx;  await\_completion(ETHERNET\_VVCT,1, RECEIVE, v\_cmd\_idx, 1 us, "Wait for receive to finish");  fetch\_result(ETHERNET\_VVCT, 1, RECEIVE, v\_cmd\_idx, **v\_data**, "Fetching result from receive operation"); |
| **ethernet\_expect ()** | **ethernet\_expect (VVCT, vvc\_instance\_idx, channel, [mac\_dest], [mac\_src], payload, msg, [alert\_level])**  The ethernet\_expect() VVC procedure adds an expect command to the Ethernet VVC executor queue, which will run as soon as all preceding commands have completed. The ethernet\_expect() procedure will perform a read operation, then check if the read data is equal to the expected data. If the read data is not equal to the expected data, an alert with severity ‘alert\_level’ will be issued. The read data will not be stored in this procedure.  Examples:  ethernet\_expect(ETHERNET\_VVCT, 1, RECEIVE, v\_mac\_dest, v\_mac\_src, v\_payload”, “Expect data from ethernet”); |

# Ignore everything beyond this point

# VVC Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record element** | **Type** | **C\_SBI\_BFM\_CONFIG\_DEFAULT** | | **Description** | |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_SBI\_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\_CMD\_QUEUE\_COUNT\_MAX | 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 initiated if 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\_sbi\_bfm\_config | C\_SBI\_BFM\_CONFIG\_DEFAULT | Configuration for SBI BFM. See quick reference for SBI BFM | |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | VVC dedicated message ID panel | |

# The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

shared\_sbi\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_sbi\_vvc\_config(1).bfm\_config.id\_for\_bfm := ID\_BFM;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_sbi\_vvc\_status record from the test sequencer. The record contents can be seen below:

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **Description** |
| current\_cmd\_idx | natural | Command index currently running |
| previous\_cmd\_idx | natural | Previous command index to run |
| pending\_cmd\_cnt | natural | Pending number of commands in the command queue |

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_sbi\_if* in order to improve readability of the code. Since the SBI interface busses can be of arbitrary size, the interface vectors have been left unconstrained. These unconstrained vectors needs to be constrained when the interface signals are instantiated. For this interface, it could look like:

signal sbi\_if\_1 : t\_sbi\_if( addr (C\_ADDR\_WIDTH-1 downto 0),

wdata(C\_DATA\_WIDTH-1 downto 0),

rdata(C\_DATA\_WIDTH-1 downto 0) );

# Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm\_vvc\_framework/doc/”.

For additional documentation on the SBI protocol, please see the SBI BFM QuickRef.

# Compilation

The SBI VVC must be compiled with VHDL 2008.   
It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.2.0 and up***
* ***UVVM VVC Framework, version 2.1.0 and up***
* ***SBI BFM***

Before compiling the SBI VVC, assure that uvvm\_vvc\_framework and uvvm\_util have been compiled.

See UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc for information about compile scripts.

**Compile order for the SBI VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_sbi | sbi\_bfm\_pkg.vhd | SBI BFM |
| bitvis\_vip\_sbi | vvc\_cmd\_pkg.vhd | SBI VVC command types and operations |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the SBI VVC library. |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | Common UVVM framework methods compiled into the SBI VVC library |
| bitvis\_vip\_sbi | vvc\_methods\_pkg.vhd | SBI VVC methods |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the SBI VVC library |
| bitvis\_vip\_sbi | sbi\_vvc.vhd | SBI VVC |

# Simulator compatibility and setup

This VVC has been compiled and tested with Modelsim version 10.3d and Riviera-PRO version 2015.10.85.

For required simulator setup see ***UVVM-Util*** Quick reference.

IMPORTANT

This is a simplified Verification IP (VIP) for SBI.

The given VIP complies with the basic SBI protocol and thus allows a normal access towards a SBI interface. This VIP is not a SBI protocol checker.

For a more advanced VIP please contact Bitvis AS at [support@bitvis.no](mailto:support@bitvis.no)

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**