**HVVC-to-VVC Bridge**

**This guide is meant for users that want to make their own HVVC-to-VVC bridge connect.**

**Users that only write test cases that are using the existing HVVCs and interfaces do NOT need to read this guide.**

# Concept

Many protocols and applications consist of several abstraction levels, e.g. physical layer, link layer, transaction layer, etc. When writing a test case for a higher level you most likely want to ignore the underlaying levels and only deal with the scope of the relevant level. The test case will be less complex and easier to both write and read. A hierarchical VVC (HVVC) is a VVC of a higher protocol level than the physical layer, i.e. it has no physical connections. The test case only communicates with the HVVC which communicate with the lower level. Data is propagated upwards and downwards between the HVVC and DUT through a standard VVC connected to the DUT.

The HVVC-to-VVC Bridge is the connection between a hierarchical VVC (HVVC) and the VVC at a lower protocol level, in this context referred to only as the VVC. Communications between the HVVC and VVC is handled by the HVVC-to-VVC Bridge. Data is transferred between the HVVC and HVVC-to-VVC Bridge on a common interface and converted in the HVVC-to-VVC Bridge to/from the specific interface of the VVC used. An example of this concept used on Ethernet is seen in Figure 1.

Figure 1 Example of HVVC-to-VVC Bridge implemented in an Ethernet HVVC.



## Interface

Communication with the bridge is done through the ports in the HVVC-to-VVC bridge. All data transfer between the HVVC and bridge is in std\_logic\_vector array format. One port is used for each direction. Data from HVVC to HVVC-to-VVC Bridge is of type t\_hvvc\_to\_bridge, and data from HVVC-to-VVC Bridge to HVVC is of type t\_bridge\_to\_hvvc.

|  |  |  |
| --- | --- | --- |
| Record ´**t\_hvvc\_to\_bridge´** | | |
| **Record element** | **Type** | **Description** |
| trigger | boolean | Trigger signal. |
| operation | t\_vvc\_operation | Operation of the VVC, e.g. RECEIVE or TRANSMIT. |
| num\_data\_words | positive | Number of data words transferred. |
| data\_words | t\_slv\_array | Data sent to the VVC. |
| dut\_if\_field\_idx | natural | Index of the interface field. |
| dut\_if\_field\_pos | t\_field\_position | Position of the interface field within the packet. |
| msg\_id\_panel | t\_msg\_id\_panel | Message ID panel of the HVVC. See section 16 of uvvm\_vvc\_framework/doc/UVVM\_VVC\_Framework\_  Essential\_Mechanisms.pdf for how to use verbosity control. |

|  |  |  |
| --- | --- | --- |
| Record ´**t\_bridge\_to\_hvvc´** | | |
| **Record element** | **Type** | **Description** |
| trigger | boolean | Trigger signal |
| data\_words | t\_slv\_array | Data received from the VVC. |

## Generic

|  |  |  |
| --- | --- | --- |
|  | | |
| **Generic element** | **Type** | **Description** |
| GC\_INSTANCE\_IDX | integer | Instance index of the VVC. |
| GC\_DUT\_IF\_FIELD\_CONFIG | t\_dut\_if\_field\_config\_direction\_array | Array of IF field configurations, see section 1.3. |
| GC\_MAX\_NUM\_WORDS | positive | Maximum number of data words transferred in one operation. |
| GC\_PHY\_MAX\_ACCESS\_TIME | time | Maximum time that the PHY interface (lowest protocol level) takes to execute an access, e.g. GMII writing 1 byte. It should account also for any margin it needs, e.g. receiver not ready. |
| GC\_SCOPE | string | Scope of the HVVC-to-VVC Bridge. |
| GC\_WORD\_ENDIANNESS | t\_word\_endianness | Word endianness when converting between different slv array widths, e.g. LOWER\_WORD\_LEFT. |
|  |  |  |

## DUT interface field configuration

If the interface of the VVC is address-based there needs to be a way to control which address to send the data to. This is done with the DUT IF field configurations. An array of t\_dut\_if\_field\_config records is defined by the user and passed to the HVVC-to-VVC Bridge through the generic of the HVVC and HVVC-to-VVC Bridge. When a transmit or receive operation is sent to the HVVC-to-VVC Bridge the index of the DUT IF field config is specified in dut\_if\_field\_idx in the hvvc\_to\_bridge port. The specified DUT IF field config states the address that shall be accessed. The address associated with each field can easily be changed by changing the DUT IF configuration.

|  |  |  |
| --- | --- | --- |
| Record ´**t\_dut\_if\_field\_config´** | | |
| **Record element** | **Type** | **Description** |
| dut\_address | unsigned | Address of the DUT IF field. |
| dut\_address\_increment | integer | Incrementation of the address on each access. |
| data\_width | positive | Width of the data per transfer, must be <= bus width. |
| use\_field | boolean | Used by the HVVC to send/request fields to/from the bridge or ignore them when not applicable (e.g. preamble in SBI). |
| field\_description | string | Description of the DUT IF field. |

# User-implementation

The bridge is implemented as an entity and is instantiated inside the HVVC. The different VVC interfaces are implemented as independent architectures due to better readability and extensibility.

## Example of implementation of GMII interface

A snippet of the implementation of GMII is shown as an example bellow.

...

-- Execute command

case hvvc\_to\_bridge.operation is

when TRANSMIT =>

-- Convert from t\_slv\_array to t\_byte\_array

v\_data\_bytes(0 to v\_num\_data\_bytes-1) := convert\_slv\_array\_to\_byte\_array(hvvc\_to\_bridge.data\_words(0 to hvvc\_to\_bridge.num\_data\_words-1), v\_byte\_endianness);

gmii\_write(GMII\_VVCT, GC\_INSTANCE\_IDX, TX, v\_data\_bytes(0 to v\_num\_data\_bytes-1), "Send data over GMII", GC\_SCOPE, hvvc\_to\_bridge.msg\_id\_panel);

v\_cmd\_idx := get\_last\_received\_cmd\_idx(GMII\_VVCT, GC\_INSTANCE\_IDX, TX, GC\_SCOPE);

await\_completion(GMII\_VVCT, GC\_INSTANCE\_IDX, TX, v\_cmd\_idx, v\_num\_transfers\*GC\_PHY\_MAX\_ACCESS\_TIME, "Wait for write to finish.", GC\_SCOPE, hvvc\_to\_bridge.msg\_id\_panel);

when RECEIVE =>

gmii\_read(GMII\_VVCT, GC\_INSTANCE\_IDX, RX, v\_num\_data\_bytes, "Read data over GMII", GC\_SCOPE, hvvc\_to\_bridge.msg\_id\_panel);

v\_cmd\_idx := get\_last\_received\_cmd\_idx(GMII\_VVCT, GC\_INSTANCE\_IDX, RX, GC\_SCOPE);

await\_completion(GMII\_VVCT, GC\_INSTANCE\_IDX, RX, v\_cmd\_idx, v\_num\_transfers\*GC\_PHY\_MAX\_ACCESS\_TIME, "Wait for read to finish.", GC\_SCOPE, hvvc\_to\_bridge.msg\_id\_panel);

fetch\_result(GMII\_VVCT, GC\_INSTANCE\_IDX, RX, v\_cmd\_idx, v\_gmii\_received\_data, "Fetching received data.", TB\_ERROR, GC\_SCOPE, hvvc\_to\_bridge.msg\_id\_panel);

-- Convert from t\_byte\_array back to t\_slv\_array

bridge\_to\_hvvc.data\_words(0 to hvvc\_to\_bridge.num\_data\_words-1) <= convert\_byte\_array\_to\_slv\_array(v\_gmii\_received\_data.data\_array(0 to v\_num\_data\_bytes-1), c\_data\_words\_width/8, v\_byte\_endianness);

when others =>

alert(TB\_ERROR, "Unsupported operation");

end case;

...

## Example of instantiation in HVVC

The example bellow shows an instantiation of the HVVC-to-VVC Bridge for GMII in an HVVC. The generics that might change in each instantiation of the HVVC, in this example the ones named GC\_\* on the right hand side of the generic map, are passed on through the HVVC from the test harness/testbench. Additional interfaces can be added by using the generate statement for each architecture.

...

gen\_hvvc\_bridge : if GC\_PHY\_INTERFACE = GMII generate

i\_hvvc\_to\_vvc\_bridge : entity bitvis\_vip\_hvvc\_to\_vvc\_bridge.hvvc\_to\_vvc\_bridge(GMII)

generic map(

GC\_INSTANCE\_IDX => GC\_PHY\_VVC\_INSTANCE\_IDX,

GC\_DUT\_IF\_FIELD\_CONFIG => GC\_DUT\_IF\_FIELD\_CONFIG,

GC\_MAX\_NUM\_WORDS => C\_MAX\_PACKET\_LENGTH,

GC\_PHY\_MAX\_ACCESS\_TIME => GC\_PHY\_MAX\_ACCESS\_TIME,

GC\_SCOPE => C\_SCOPE

)

port map(

hvvc\_to\_bridge => hvvc\_to\_bridge,

bridge\_to\_hvvc => bridge\_to\_hvvc

);

end generate gen\_hvvc\_bridge;

...

# Procedures

The following procedures are used by the HVVC when transmitting or requesting data from the HVVC-to-VVC Bridge.

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **blocking\_send\_to\_bridge()** | **blocking\_send\_to\_bridge(hvvc\_to\_bridge, bridge\_to\_hvvc, data\_words, dut\_if\_field\_idx, dut\_if\_field\_pos, scope, msg\_id\_panel)**  Sends a data array to the HVVC-to-VVC Bridge and waits for a trigger signalling the bridge has finished. This procedure blocks the sequencer until it is done.  Examples:  blocking\_send\_to\_bridge(hvvc\_to\_bridge, bridge\_to\_hvvc, v\_data\_array(0 to 9), C\_FIELD\_IDX\_PAYLOAD, FIRST, C\_SCOPE,  v\_msg\_id\_panel); |
| **blocking\_request\_from\_bridge()** | **blocking\_request\_from\_bridge(hvvc\_to\_bridge, bridge\_to\_hvvc, num\_data\_words, dut\_if\_field\_idx, dut\_if\_field\_pos, scope, msg\_id\_panel)**  Requests data from the HVVC-to-VVC Bridge and waits for a trigger signalling the bridge has finished. This procedure blocks the sequencer until it is done.  Examples:  blocking\_request\_from\_bridge(hvvc\_to\_bridge, bridge\_to\_hvvc, 10, C\_FIELD\_IDX\_PAYLOAD, LAST, C\_SCOPE, v\_msg\_id\_panel);  v\_receive\_words := bridge\_to\_hvvc.data\_words(0 to 9); -- Save the received data |
|  |  |

# Additional Documentation

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

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**