**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 as the sub-VVC. Communications between the HVVC and sub-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 sub-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

Communications with the bridge is done through the ports in the HVVC-to-VVC bridge. All data transfer between the HVVC and bridge is in byte 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\_sub\_vvc\_operation | The operation of the sub VVC, e.g. RECEIVE or TRANSMIT |
| num\_data\_bytes | positive | The number of bytes of data that is transferred. |
| data\_bytes | t\_byte\_array | Data sent to sub VVC. |
| dut\_if\_field\_idx | integer | The index of the interface field. |
| current\_byte\_idx\_in\_field | natural | The byte number in the interface field. |
| msg\_id\_panel | t\_msg\_id\_panel | Message ID panel |

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

## Generic

|  |  |  |
| --- | --- | --- |
|  | | |
| **Generic element** | **Type** | **Description** |
| GC\_INTERFACE | t\_interface | The interface of the sub-VVC. |
| GC\_INSTANCE\_IDX | integer | Instance index of the sub-VVC. |
| GC\_CHANNEL | t\_channel | Channel of the sub-VVC. |
| GC\_DUT\_IF\_FIELD\_CONFIG | t\_dut\_if\_field\_config\_channel\_array | Array of IF field configurations. |
| GC\_SCOPE | string | Scope of the HVVC-to-VVC Bridge. |

## DUT IF field configuration

If the interface of the sub-VVC is address-based there need 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. |
| 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 interfaces are implemented in a case statement in the architecture. New sub-VVC interfaces are added here. The implementation of SBI is shown as an example bellow. For non-address-based interfaces v\_dut\_address is ignored.

## Example

...

when SBI =>

case hvvc\_to\_bridge.operation is

when TRANSMIT =>

-- Loop through bytes

for i in 0 to hvvc\_to\_bridge.num\_data\_bytes-1 loop

-- Send data over SBI

sbi\_write(SBI\_VVCT, GC\_INSTANCE\_IDX, v\_dut\_address, hvvc\_to\_bridge.data\_bytes(i), "Send data over SBI",

GC\_SCOPE, USE\_PROVIDED\_MSG\_ID\_PANEL, hvvc\_to\_bridge.msg\_id\_panel);

v\_dut\_address := v\_dut\_address + v\_dut\_address\_increment;

end loop;

when RECEIVE =>

-- Loop through bytes

for i in 0 to hvvc\_to\_bridge.num\_data\_bytes-1 loop

-- Read data over SBI

sbi\_read(SBI\_VVCT, GC\_INSTANCE\_IDX, v\_dut\_address, "Read data over SBI",

GC\_SCOPE, USE\_PROVIDED\_MSG\_ID\_PANEL, hvvc\_to\_bridge.msg\_id\_panel);

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

await\_completion(SBI\_VVCT, GC\_INSTANCE\_IDX, v\_cmd\_idx, 1 ms, "Wait for read to finish.",

GC\_SCOPE, USE\_PROVIDED\_MSG\_ID\_PANEL, hvvc\_to\_bridge.msg\_id\_panel);

fetch\_result(SBI\_VVCT, GC\_INSTANCE\_IDX, v\_cmd\_idx, v\_sbi\_received\_data, "Fetching received data.",

TB\_ERROR, GC\_SCOPE, USE\_PROVIDED\_MSG\_ID\_PANEL, hvvc\_to\_bridge.msg\_id\_panel);

bridge\_to\_hvvc.data\_bytes(i) <= v\_sbi\_received\_data(7 downto 0);

v\_dut\_address := v\_dut\_address + v\_dut\_address\_increment;

end loop;

end case;

...

# Procedures

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

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **hvvc\_to\_bridge\_trigger()** | **hvvc\_to\_bridge\_trigger(hvvc\_to\_bridge)**  The hvvc\_to\_bridge\_trigger () procedure generates a trigger pulse on the trigger in the hvvc\_to\_bridge record. |
| **send\_to\_sub()** | **send\_to\_sub(hvvc\_to\_bridge, operation, data\_bytes, dut\_if\_field\_idx, current\_byte\_idx\_in\_field, msg\_id\_panel)**  **send\_to\_sub(hvvc\_to\_bridge, operation, num\_data\_bytes, dut\_if\_field\_idx, current\_byte\_idx\_in\_field, msg\_id\_panel)**  Sends operation to HVVC-to-VVC Bridge.  Examples:  send\_to\_sub(hvvc\_to\_bridge, TRANSMIT, v\_transmit\_bytes, 0, 0, v\_msg\_id\_panel); -- Transmit byte array v\_transmit\_bytes  send\_to\_sub(hvvc\_to\_bridge, RECEIVE, 10, 0, 0, v\_msg\_id\_panel); -- Receive 10 bytes |
| **blocking\_send\_to\_sub()** | **blocking\_send\_to\_sub(hvvc\_to\_bridge, bridge\_to\_hvvc, operation, data\_bytes, dut\_if\_field\_idx, current\_byte\_idx\_in\_field, msg\_id\_panel)**  **blocking\_send\_to\_sub(hvvc\_to\_bridge, bridge\_to\_hvvc, operation, num\_data\_bytes, dut\_if\_field\_idx, current\_byte\_idx\_in\_field, msg\_id\_panel)**  Sends operation to HVVC-to-VVC Bridge and awaits trigger.  Examples:  blocking\_send\_to\_sub(hvvc\_to\_bridge, bridge\_to\_hvvc, TRANSMIT, v\_transmit\_bytes, 0, 0, v\_msg\_id\_panel); -- Transmit byte array  -- v\_transmit\_bytes  blocking\_send\_to\_sub(hvvc\_to\_bridge, bridge\_to\_hvvc, RECEIVE, 10, 0, 0, v\_msg\_id\_panel); -- Receive 10 bytes  v\_receive\_bytes := bridge\_to\_hvvc.data\_bytes(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**