

# 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.

## 1 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 underlying 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.

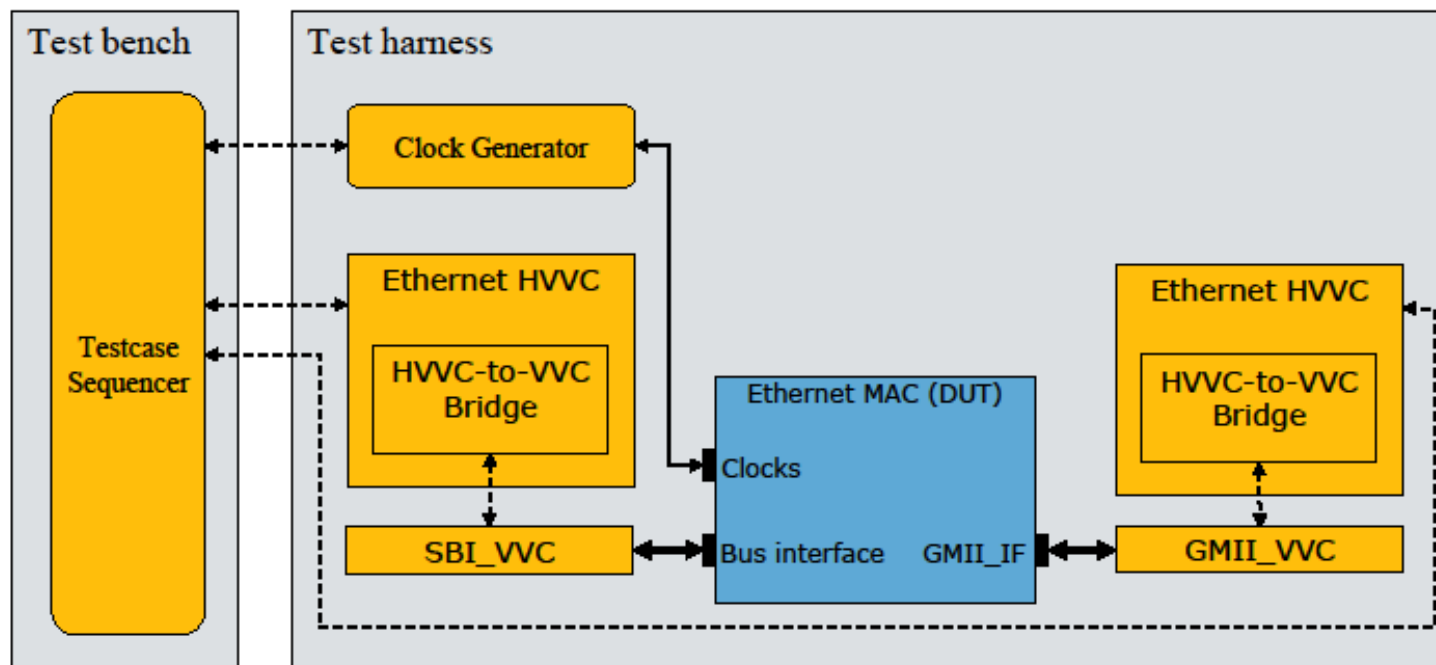


Figure 1: Example of HVVC-to-VVC Bridge implemented in an Ethernet HVVC.

## 1.1 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 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 the 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 the VVC.

## 1.2 Generic

Generic element	Type	Description
GC_INTERFACE	t_interface	The interface of the VVC.
GC_INSTANCE_IDX	integer	Instance index of the VVC.
GC_CHANNEL	t_channel	Channel of the 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.

### 1.3 DUT interface field configuration

If the interface of the 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 `hvv_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
<code>dut_address</code>	unsigned	Address of the DUT IF field.
<code>dut_address_increment</code>	integer	Incrementation of the address on each access.
<code>field_description</code>	string	Description of the DUT IF field.

## 2 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 VVC interfaces are added here.

### 2.1 Example of implementation of SBI interface

The implementation of SBI is shown as an example bellow. For non-address-based interfaces `v_dut_address` is ignored.

```
...

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;

  ...
```

## 2.2 Example of instantiation in HVVC

The example bellow shows an instantiation of the HVVC-to-VVC Bridge 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/test bench.

```
...

i_hvvc_to_vvc_bridge : entity bitvis_vip_hvvc_to_vvc_bridge.hvvc_to_vvc_bridge
  generic map(
    GC_INTERFACE          => GC_INTERFACE,
    GC_INSTANCE_IDX       => GC_VVC_INSTANCE_IDX,
    GC_CHANNEL             => C_CHANNEL,
    GC_DUT_IF_FIELD_CONFIG => GC_DUT_IF_FIELD_CONFIG,
    GC_MAX_NUM_BYTES       => C_MAX_PACKET_LENGTH,
    GC_SCOPE               => C_SCOPE
  )
  port map(
    hvvc_to_bridge => hvvc_to_bridge,
    bridge_to_hvvc => bridge_to_hvvc
  );

...
```

### 3 Procedures

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

Procedure	Description
<b>hvv_c_to_bridge_trigger()</b>	<p><b>hvv_c_to_bridge_trigger(hvv_c_to_bridge)</b></p> <p>The hvvc_to_bridge_trigger () procedure generates a trigger pulse on the trigger in the hvvc_to_bridge record.</p>
<b>send_to_bridge()</b>	<p><b>send_to_bridge(hvv_c_to_bridge, operation, data_bytes, dut_if_field_idx, current_byte_idx_in_field, msg_id_panel)</b>  <b>send_to_bridge(hvv_c_to_bridge, operation, num_data_bytes, dut_if_field_idx, current_byte_idx_in_field, msg_id_panel)</b></p> <p>Sends operation to HVVC-to-VVC Bridge.</p> <p>Examples:</p> <pre>send_to_bridge(hvv_c_to_bridge, TRANSMIT, v_transmit_bytes, 0, 0, v_msg_id_panel); -- Transmit byte array v_transmit_bytes send_to_bridge(hvv_c_to_bridge, RECEIVE, 10, 0, 0, v_msg_id_panel); -- Receive 10 bytes</pre>
<b>blocking_send_to_bridge()</b>	<p><b>blocking_send_to_bridge(hvv_c_to_bridge, bridge_to_hvv_c, operation, data_bytes, dut_if_field_idx, current_byte_idx_in_field, msg_id_panel)</b>  <b>blocking_send_to_bridge(hvv_c_to_bridge, bridge_to_hvv_c, operation, num_data_bytes, dut_if_field_idx, current_byte_idx_in_field, msg_id_panel)</b></p> <p>Sends operation to HVVC-to-VVC Bridge and awaits trigger.</p> <p>Examples:</p> <pre>-- Transmit byte array v_transmit_bytes blocking_send_to_bridge(hvv_c_to_bridge, bridge_to_hvv_c, TRANSMIT, v_transmit_bytes, 0, 0, v_msg_id_panel); blocking_send_to_bridge(hvv_c_to_bridge, bridge_to_hvv_c, RECEIVE, 10, 0, 0, v_msg_id_panel); -- Receive 10 bytes v_receive_bytes := bridge_to_hvv_c.data_bytes(0 to 9); -- Save the received data</pre>

## 4 Additional Documentation

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

### INTELLECTUAL PROPERTY

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.