Design specification for the DUT used in the "Formal verification part-0" practice session

1 Functional description

Functional diagram of the DUT is shown in Figure 1.

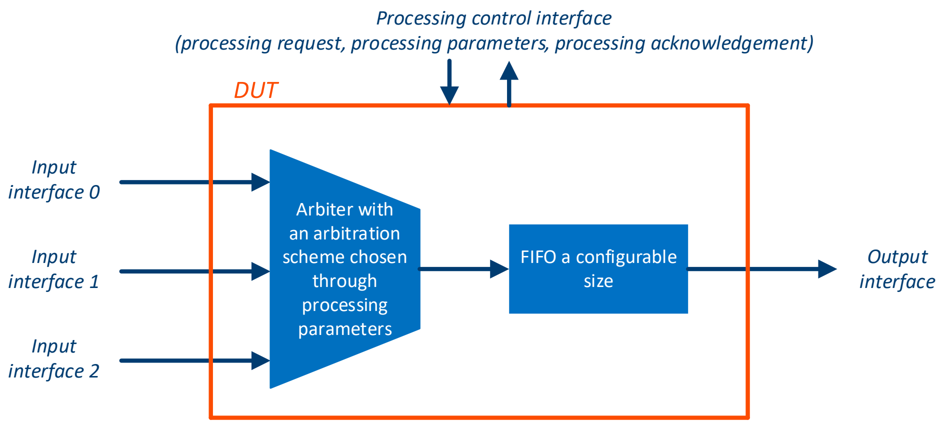

Processing control interface
(processing request, processing parameters, processing acknowledgement)
DUT
Input interface 0
Input interface 1
Input interface 2
Arbiter with an arbitration scheme chosen through processing parameters
FIFO a configurable size
Output interface

Figure 1. Functional diagram of the DUT

Processing within the DUT is controlled through the "processing control interface" (described in section 3.3. After processing is requested through the interface:

- The DUT needs to receive all valid input data (all valid data until data indicated as last data) from all enabled "input interfaces". More details about the "input interfaces" can be found in section 3.4. Enabling of input interfaces is driven through the processing control interface (go to section 3.3 for more details)
- The DUT needs to send all the data from all enabled input interfaces out through the "output interface", described in section 3.5
- When all the data is sent out, the DUT needs to acknowledge the processing request through the processing control interface

Such a procedure as described above (from a processing request to a processing acknowledgment) is named as a "frame" in later parts of this document. After reset, the DUT can process several frames, one by one, as shown in

Figure 2. A processing request is noted there as "proc_req", whereas a processing acknowledgment – as "proc_ack".

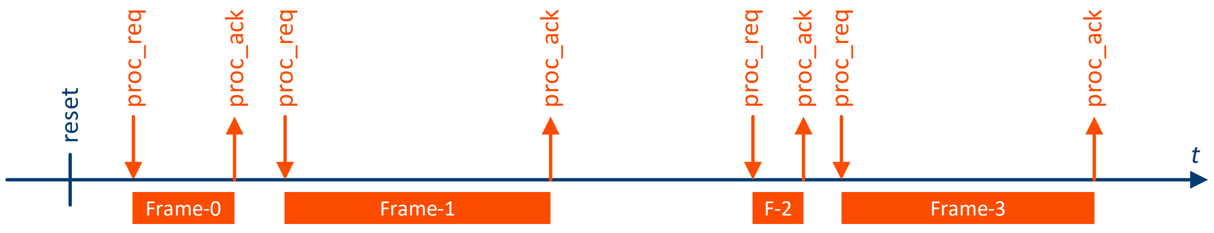


Figure 2. Frame-by-frame processing concept

Functional chart of the DUT is shown in Figure 3. As “transferring”, a full transfer from the input to the output interface is noted there. As “last data transferred” - sending of last data out through the output interface.

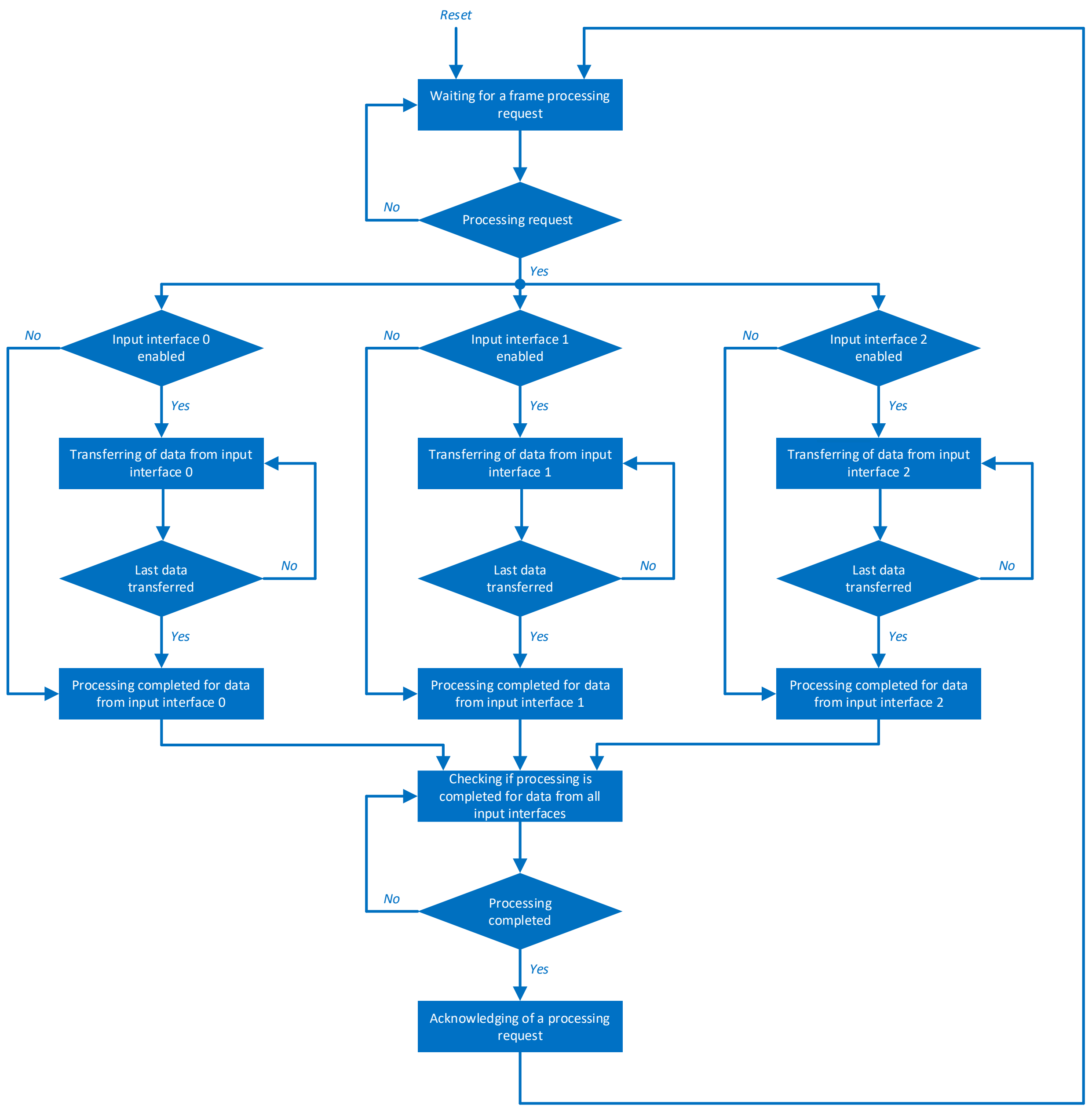
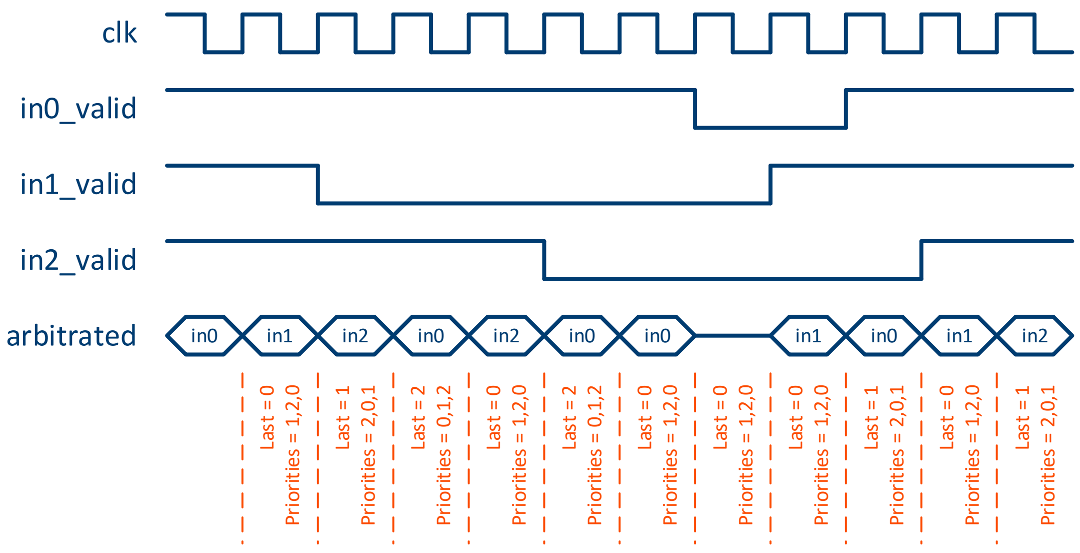

Reset
Waiting for a frame processing request
Processing request
No
Yes
Input interface 0 enabled
Input interface 1 enabled
Input interface 2 enabled
No
Yes
Transferring of data from input interface 0
Transferring of data from input interface 1
Transferring of data from input interface 2
Last data transferred
No
Yes
Processing completed for data from input interface 0
Processing completed for data from input interface 1
Processing completed for data from input interface 2
Checking if processing is completed for data from all input interfaces
Processing completed
No
Yes
Acknowledging of a processing request

Figure 3. Processing scheme of the DUT

Arbitration of data from all enabled input interfaces happens based on an “arbitration mode”, provided through the processing control interface (see section 3.3 for more details). Current DUT supports only one arbitration mode – Round-Robin arbitration. If any other mode is provided to the current DUT, the DUT may behave in an unpredictable way.

The Round-Robin algorithm is shown in

Figure 4. General idea of that algorithm's implementation in hardware is that the arbiter arbitrates a first interface with valid data after a recently arbitrated one.



clk
in0_valid
in1_valid
in2_valid
arbitrated
in0
in1
in2
in0
in2
in0
in0
in1
in0
in1
in2
Last = 0 Priorities = 1,2,0
Last = 1 Priorities = 2,0,1
Last = 2 Priorities = 0,1,2
Last = 0 Priorities = 1,2,0
Last = 2 Priorities = 0,1,2
Last = 0 Priorities = 1,2,0
Last = 0 Priorities = 1,2,0
Last = 0 Priorities = 1,2,0
Last = 1 Priorities = 2,0,1
Last = 0 Priorities = 1,2,0
Last = 1 Priorities = 2,0,1

Figure 4. Round-Robin arbitration scheme

As visible in the functional diagram from Figure 1, the DUT is also expected to implement a FIFO of a configurable size. The FIFO is needed to reduce latencies in a system implementing the DUT. A system integrator should be able to configure number of entries within the FIFO through a top-level parameter of the DUT.

Width (number of bits) of data transferred through the DUT should be also configurable through a top-level parameter of the DUT.

2 Microarchitecture description

All the RTL code related to the DUT can be found in a following file:
<..>/intel_formal_verification_practice_session_0/rtl/dut_toplevel.sv

The file is structured as shown in the microarchitecture diagram of the DUT from

Figure 5. The diagram simply reflects all the DUT's functionalities described in section 1.

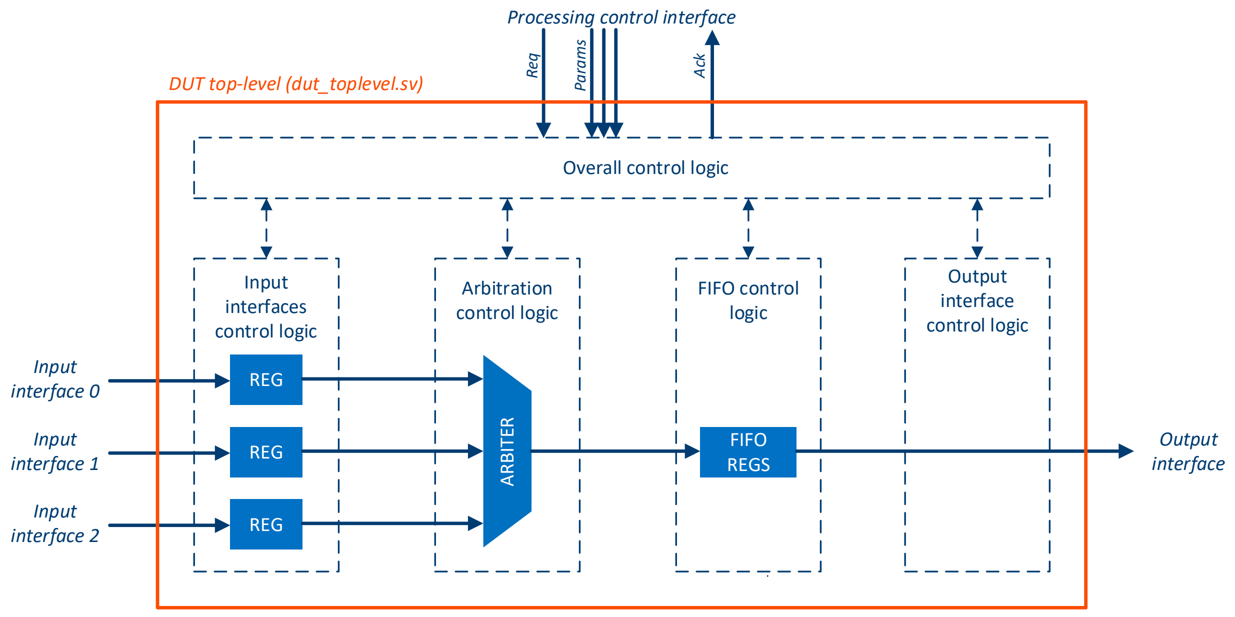

Processing control interface
Req
Params
Ack
DUT top-level (dut_toplevel.sv)
Overall control logic
Input interfaces control logic
Arbitration control logic
FIFO control logic
Output interface control logic
Input interface 0
Input interface 1
Input interface 2
REG
REG
REG
ARBITER
FIFO REGS
Output interface

Figure 5. Microarchitecture diagram of the DUT

Figure 5 presents all data path registers. These are in the input interfaces control logic and in the FIFO.

The DUT is in a single clock domain and has a single clock input, named “clk”.

The DUT is in a single power domain and has a single reset input, named “nreset”. That reset input is active low (when that signal low, then reset is asserted). The reset is asserted asynchronously and deasserted synchronously (synchronously to the ‘clk’ clock input).

Configurability of the DUT (described in section 1) is provided through following parameters:

- DATA_WIDTH – width (number of bits) of data transferred through the DUT
- FIFO_HEIGHT – number of entries within the FIFO implemented in the DUT

3 Interfaces

3.1 Standard request/acknowledgement interface

A standard request/acknowledgement interface consists of signals shown in Table 1. This interface is used by an initiator to request any action from a target, which then requires confirmation (acknowledgment) send by the target back to the initiator. Both request and acknowledgement can be optionally equipped with associated data. Data transfers happen when both ‘req’ and ‘ack’ signals are high in the same clock cycle.

Table 1. Signals of a standard request/acknowledgment interface

Signal name	Width	Direction	Description
req	Single-bit signal	Initiator -> target	**<u>Request</u>** • This signal can go high only, when acknowledgement ('ack' signal) was low in a previous cycle • This signal needs to be kept high until acknowledgement ('ack' signal) goes high • After acknowledgement ('ack' signal) goes high, this signal ever goes low • When this signal is high, it indicates that there is valid data exposed on the 'req_data' signal
req_data	Multi-bit signal	Initiator -> target	**<u>Request data</u>** • This signal is optional • Request data can consist of several signals, which are transmitted together through the interface • After exposing to the interface (together with the 'req' signal), data needs to be stable (unchanged) in all cycles when the request ('req' signal) is high
ack	Single-bit signal	Target -> initiator	**<u>Acknowledgement</u>** • This signal can go high only when request ('req' signal) is high • After request ('req' signal) goes high, this signal goes ever high • This signal needs to be kept high at least while request ('req' signal) is high • After request ('req' signal) goes low, this signal goes ever low • When this signal is high, it indicates that there is valid data exposed on the 'ack_data' signal
ack_data	Multi-bit signal	Target -> initiator	**<u>Acknowledgement data</u>** • This signal is optional • Acknowledgement data can consist of several signals, which are transmitted together through the interface • After exposing to the interface (together with the 'ack' signal), data needs to be stable (unchanged) in all cycles when the acknowledgement ('ack' signal) is high

An example waveform of a standard request/acknowledgment interface is shown in Figure 6.

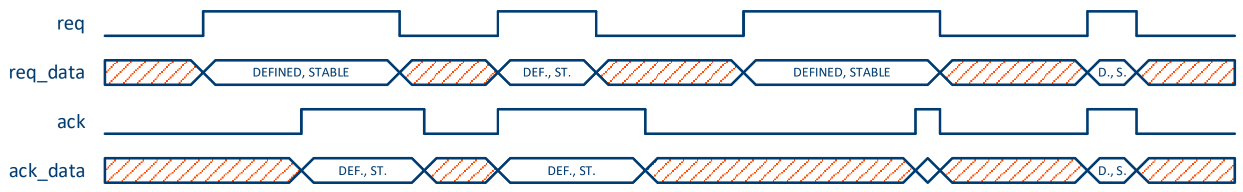


Figure 6. An example waveform of a standard request/acknowledgement interface

3.2 Standard valid/ready interface

A standard valid/ready interface consists of signals shown in Table 2. This interface is used to transfer data from an initiator to a target. Data transfer happens when both 'valid' and 'ready' signals are high in the same clock cycle.

Table 2. Signals of a standard valid/ready interface

Signal name	Width	Direction	Description
valid	Single-bit signal	Initiator -> target	**Valid flag** • This signal can go high independently on the 'ready' signal (can go high earlier, later or at the same time as the 'ready' signal) • After going high, this signal needs to be kept high until data is transmitted (until ready is high) • When this signal is high, it indicates that there is valid data exposed on the 'data' signal by an initiator
ready	Single-bit signal	Target -> initiator	**Ready flag** • When high, it indicates that a target can consume data • This signal can go high independently on the 'valid' signal (can go high earlier, later or at the same time as the 'valid' signal) • When this signal is not high when 'valid' is high, then it ever goes high (each transfer must be completed)
data	Multi-bit signal	Initiator -> target	**Data** • Data can consist of several signals, which are transmitted together through the interface • After exposing to the interface (together with the 'valid' signal), data needs to be stable (unchanged) until the data is transmitted (until 'ready' is high)

An example waveform of a standard valid/ready interface is shown in
Figure 7.

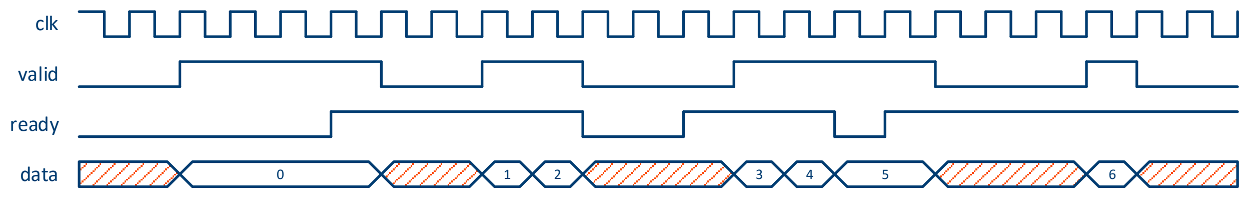


Figure 7. An example waveform of a standard valid/ready interface

3.3 Processing control interface

The processing control interface is based on a standard request/acknowledgment interface, described in section 3.1. All signals from that interface, together with additional assumptions related to the signals, are shown in Table 3.

Table 3. Signals of the processing control interface

Signal name	Width [bits]	DUT port direction	Description
proc_req	1	input	**<u>Processing request</u>** • Behaves as a 'req' signal from a standard request/acknowledgement interface • When this signal is low and there is any valid input data (when any 'in{0,1,2}_valid' signal is high), then this signal ever goes high
proc_req_in{0,1,2}_en	1 (each)	input	**<u>Enable flag for a corresponding input interface</u>** • Behaves as a 'data' signal from a standard request/acknowledgement interface
proc_req_in{0,1,2}_ arb_mode_id	1 (each)	input	**<u>Arbitration mode ID for a corresponding input interface</u>** • Behaves as a 'data' signal from a standard request/acknowledgement interface • In a current DUT's version, needs to be '1'b0' (Round-Robin arbitration scheme)
proc_ack	1	output	**<u>Processing acknowledgement</u>** • Behaves as an 'ack' signal from a standard request/acknowledgement interface • This signal can go high only when all data from a current frame has been sent out through the output interface (only when a transfer with the 'out_data_last' signal high has happened)

3.4 Input interfaces

There are three input interfaces. All of them work based on a standard valid/ready interface, described in section 3.2, with some additional assumptions. Names of signals from the interfaces start from prefixes indicating a given interface – ‘in0_’ for input interface 0, ‘in1_’ for input interface 1 and ‘in2_’ for input interface 2.

All signals of the input interfaces, together with additional assumptions related to the signals, are shown in Table 4.

Table 4. Signals of the input interfaces

Signal name	Width [bits]	DUT port direction	Description
in{0,1,2}_valid	1	input	**Valid flag** • Behaves as a ‘valid’ signal from a standard valid/ready interface • When processing is requested (when ‘proc_req’ signal from the processing control interface is high), a corresponding input interface is enabled (‘proc_req_in{0,1,2}_en’ signal from the processing control interface is high) and a last transfer from a corresponding input interface has not been exposed yet for a given frame (when there has not been ‘in{0,1,2}_valid’ high with in{0,1,2}_data_last’ high in a given frame), then this signal is ever high again
in{0,1,2}_ready	1	output	**Ready flag** • Behaves as a ‘ready’ signal from a standard valid/ready interface • Can be high only, when a frame processing is active (when the ‘proc_req’ signal from the processing control interface is high and the ‘proc_ack’ signal from that interface is low) • Can be high only, when a corresponding interface is enabled (when a corresponding ‘proc_req_in{0,1,2}_en’ signal from the processing control interface is high) • Can be high only if a last transfer from a corresponding input interface has not happened yet for a given frame (when a transfer with a corresponding signal ‘in{0,1,2}_data_last’ high has not happened yet in a given frame)

Signal name	Width [bits]	DUT port direction	Description
in{0,1,2}_data	DATA_WIDTH	input	**<u>Data</u>** • Behaves as a 'data' signal from a standard valid/ready interface
in{0,1,2}_data_last	1	input	**<u>Indicator of last data in a frame</u>** • Behaves as 'data' from a standard valid/ready interface • Indicates a last transfer (last data) in a given frame • This signal is ever high, if there have been any transfers with that flag being low (from a corresponding input interface) • Can be high even for first data in a given frame (it would mean only one transfer per a frame)

3.5 Output interface

The output interface is based on a standard valid/ready interface, described in section 3.2. All signals from that interface, together with additional assumptions related to the signals, are shown in Table 5.

Table 5. Signals of the output interface

Signal name	Width [bits]	DUT port direction	Description
out_valid	1	output	**<u>Valid flag</u>** • Behaves as a 'valid' signal from a standard valid/ready interface • Can be high only, when a frame processing is active (when the 'proc_req' signal from the processing control interface is high and the 'proc_ack' signal from that interface is low)
out_ready	1	input	**<u>Ready flag</u>** • Behaves as 'ready 'for a standard valid/ready interface
out_data	DATA_WIDTH	output	**<u>Data</u>** • Behaves as a 'data' signal from a standard valid/ready interface • Must be the same as corresponding data received through a corresponding input interface

Signal name	Width [bits]	DUT port direction	Description
out_data_source_id	2	output	**<u>Source ID</u>** • Behaves as a 'data' signal from a standard valid/ready interface • This signal indicates from which input interface the data come: • 2'b00 – from input interface 0 • 2'b01 – from input interface 1 • 2'b10 – from input interface 2
out_data_last	1	output	**<u>Indicator of last data in a frame</u>** • Behaves as a 'data' signal from a standard valid/ready interface • This signal indicates a last output transfer (last data) in a given frame (last at all, taking all input interfaces into account) • This signal needs to be high in each frame • This signal Can be high only for a last output transfer

4 SVA properties

SVA properties for the DUT can be found in following locations:

- Interface properties: *<..>/intel_formal_verification_practice_session_0/properties/dut_toplevel_interface_properties.sv*
- Properties related to internal signals: *<..>/intel_formal_verification_practice_session_0/properties/dut_toplevel_internal_properties.sv*