VC Verification IP
AMBA AXI
FAQ

Version O-2018.12, December 2018

SYNOPSYS®

Contents

1 AMBA SVT Frequently Asked Questions (FAQ)

This section provides the following list of frequently asked general questions for VC VIP Interlaken:

1.1 General

1.1.1 How do I disable reporting of svt_axi_system_configuration while using AMBA VIP's with UVM?

Note
This printing information is useful for debugging. If you have disabled the printing information in your simulation then enable this before sharing the simulation log with Synopsys support. It is recommended to keep this configuration reporting ON (default).

In order to disable this feature in your simulation :: >>`uvm_test_top.env.axi_system_env [connect_ph]`

```
**************System Configuration*************
```

Use runtime switch

```
+uvm_set_action=uvm_test_top.env.\*,connect_ph,UVM_INFO,UVM_NO_ACTION
```

Specifying hierarchical path `uvm_test_top.env.\*` will switch off all activities in connect_ph either from VIP(Synopsys) or DUT (Customer).

Note
For specific disabling of reporting, specify the exact path after `uvm_test_top.env`.

1.1.2 How do I set the virtual interface in AXI VIP?

You can set the virtual interface in the following ways:

1. Add the following to `top.sv`:

   ```
   svt_axi_if  axi_if();
    initial begin
      uvm_config_db#(virtual svt_axi_if)::set(uvm_root::get(),
      "uvm_test_top.env.axi_system_env", "vif", axi_if);
    end
   ```

2. By using the method, `set_if()` provided in the `axi_svt_uvm_system_configuration.sv`
   `this.set_if(test_top.axi_if);`

1.2 Configuration

1.2.1 How do I configure different data widths for different AXI STREAM VIP components?

For use case, consider a signal TDATA.

The SVT_AXI_MAX_TDATA_WIDTH macro defines the *maximum* width (maximum footprint) of the tdata signal across the complete system (all masters, slaves). The default value is 128.

In order to change the default value from 128 to 256, create a `svt_axi_user_defines.svi` file with the following content:

```
=====================
`define SVT_AXI_USER_DEFINES_SVI
`define SVT_AXI_MAX_TDATA_WIDTH 256
```

=====================

It is also required to pass the define SVT_AXI_INCLUDE_USER_DEFINES as command-line argument (for example: +define+SVT_AXI_INCLUDE_USER_DEFINES).

The tdata width of the individual master and slave components can be controlled using the port configuration attribute tdata_width. The tdata_width must be less than or equal to SVT_AXI_MAX_TDATA_WIDTH. The VIP drives only that part of tdata signal specified by tdata_width.

1.3 Usage

1.3.1 How do I connect DUT and AXI VIP having different widths?

Consider a scenario where it is required to verify a master DUT having IF width of 32 with slave VIP having IF width of 64.

Connect the master DUT's 32-bit IF to the lower 32-bits of VIP slaveIF only. (Do not connect the upper 32-bits)

1.3.2 How do I generate data interleave response for read transactions in AXI?

In order to generate the read interleave transactions for AXI3, follow the following rules:

- Apply the constraints on the slave transaction object in order to enable transaction interleave (enable_interleave field in the transaction object).
- Send the read transactions with multiple beats.

Enable the AXI3 Interleave transactions

In order to enable interleave transactions, it is required to apply the following constraints in the slave transaction object. In this example, VIP interconnect is used, hence it is required to apply the constraints on the `svt_axi_ic_slave_transaction` as per the following description:

```
class cust_svt_axi_ic_slave_transaction extends svt_axi_ic_slave_transaction;
// Declare user-defined constraints over here
constraint slave_ic_constraints
{
    enable_interleave == 1;
    addr_ready_delay == 0; ……
}
…….
…….
endclass: cust_svt_axi_ic_slave_transaction
```

In the test do factory override

```
set_type_override_by_type(svt_axi_ic_slave_transaction::get_type(),cust_axi_ic_slave_tr
ansaction::get_type());
```

Note
The extended class above set the enable_interleave to 1 in order to enable the read transaction interleaves.

1.3.3 How do I generate out of order read data from AXI slave VIP?

Set the following configuration attribute in the customer configuration file extended from the `svt_axi_system_configuration` file.

Example:

```
class cust_axi_system_configuration extends svt_axi_system_configuration;
…….
this.slave_cfg[0].reordering_algorithm=svt_axi_port_configuration::
RANDOM;
……..
endclass
```

Note
Ensure that the rready is getting delayed for at least one cycle.

1.3.4 How do I specify user-defined constraints?

There are two steps to achieve this:

1. Extend the user defined transaction class from the master or slave transaction class.

   Example:

   ```
   class cust_axi_snoop_transaction extends svt_axi_master_snoop_transaction;
          ………………..
          constraint my_constraint {
                data_before_resp == 1;
                acready_delay== 0;
                reference_event_for_acready_delay == ACVALID;
                reference_event_for_first_cdvalid_delay ==
                SNOOP_ADDR_HANDSHAKE;
                foreach(cdvalid_delay[i])
                cdvalid_delay[i] ==0;
                reference_event_for_next_cdvalid_delay ==
                PREV_SNOOP_DATA_HANDSHAKE;
                crvalid_delay ==10;
                reference_event_for_crvalid_delay ==
                SNOOP_ADDR_HANDSHAKE;}
              .………………..
   endclass
   ```

2. Do the following factory override:

   ```
   set_type_override_by_type(svt_axi_master_snoop_transaction::get_type(),
   cust_axi_snoop_transaction::get_type());
   ```

1.3.5 How do I drive channel signals low during idle cycles in AXI VIP?

Set the following configuration attribute in the customer configuration file extended from `svt_axi_system_configuration` file.

Example:

```
class cust_axi_system_configuration extends svt_axi_system_configuration;
…….
this.slave_cfg[0].read_data_chan_idle_val=svt_axi_port_configuration::
INACTIVE_CHAN_LOW_VAL;
……..
endclass
```

Note
The same can be applied to the remaining channels depending on the usage of master or slave VIP.

1.3.6 How do I enable system monitor data integrity check on passive slave?

By default, the system monitor is in passive mode.

Update the content of the shadow memory with the return data from the read transaction. However in order to get the system monitor not to update the shadow memory with the read return data, configure the attribute as:

```
svt_axi_port_configuration::memory_update_for_read_xact_enable = 0
```

Passive slave memory needs to be aware of the backdoor writes to memory. Setting this attribute in configuration allows passive slave memory to be updated according to RDATA seen in the transaction coming from the slave. Note that the passive slave memory is updated when all read data beats have been transmitted by slave and accepted by the master. For an EXCLUSIVE transaction memory is updated only when all beats in a transaction have an EXOKAY response. For a normal transaction, memory is updated only when all the beats in a transaction have an OKAY response.

1.3.7 How to send dvm or barrier transactions such that the IDs of those transactions do not overlap with outstanding active non-dvm or non-barrier transactions?

There are two different ways to achieve this. The process is same for both DVM and Barrier. First, ID overlap can be disabled in the port configuration. This way when transactions are randomized both DVM and non-DVM, or barrier and non-barrier will not have overlapped IDs. Each Master can be enabled to have ID overlap for DVM or Barrier in the port configuration. The `dvm_id_overlap` and `barrier_id_overlap` are configuration parameters. Refer AXI Class Reference Documents for further details.

Secondly, if ID overlap is enabled by setting `dvm_id_overlap` = 1 or `barrier_id_overlap` = 1 for DVM and barrier transactions respectively, then method `svt_axi_master::get_ids_used_by_active_master_transactions()` can be used to gather the list of IDs used by non-DVM or non-barrier transactions and while randomizing the sequence_items before sending to the driver a single line of constraint can be added to ensure transaction ID does not match any element of the ID list obtained from that function. The following is the example to illustrate the usage. For more details, see svt_axi_master section in the AXI Class Reference Documents (

Example: `dvm_id_overlap` and `barrier_id_overlap` both are set to '1' for Master[0]. Now, before sequence_items are randomized within the body() of the sequence following steps are executed.

```
svt_configuration base_cfg;
    uvm or ovm_component my_component;
    p_sequencer.get_cfg(base_cfg);
    if (!$cast(sys_cfg, base_cfg)) begin
      `svt_xvm_fatal("body", "Unable to $cast the configuration to a
svt_axi_system_configuration class");
    end
    my_component = p_sequencer.get_parent();
    if (!$cast(my_system_env,my_component)) begin
      `svt_xvm_fatal("body", "Expected parent of svt_axi_system_sequencer to be of type
svt_axi_system_env, but it is not");
    end

    while(1) begin

if(my_system_env.master[port_id].driver.get_ids_used_by_active_master_transactions(id_q
,"non_dvm",0)) begin
                  …….. custom logic ….
                 <create sequence item>;
                <randomize sequence item with { ….custom constraints….;  if(id_q.size()
> 0) { id inside {id_q}; }  };  >
                  …….. custom logic ….
                 <send sequence item>;
                  …….. custom logic ….
// dvm transaction sent so break the while() loop
break;
    end
    else begin
        <save number of outstanding transactions to "var1">;
        <if(var1 > 0) wait until number of outstanding transactions reduces by at least
one i.e. < current value "var1">;
    end
    end // while()
```

In order to get number of outstanding transactions either transactions sent from the sequence can be tracked within the sequence and a queue of outstanding transactions are maintained or VIP built-in method `my_system_env.master[port_id].driver. get_number_of_outstanding_master_transactions()` can be used for this purpose. For more details, see “svt_axi_master” section in theAXI Class Reference Documents.

The previous sequence can be referred from "svt_axi_ace_master_multipart_dvm_virtual_sequence" in AXI Class Reference Document under sequence section.

```
/*Returns the number of outstanding transactions */
function int get_number_of_outstanding_master_transactions(bit silent = 1);
```

1.4 Debug

1.4.1 How do I resolve error: Response Queue Overflow in UVM?

AXI VIP master driver returns the response for every completed transaction (this behavior cannot be changed). All these responses get queued up in the response queue. The default depth of the queue is eight.

This error is displayed when the response overflow occurs and can happen in the following situations:

1. When the master sequence does not retrieve the responses, for example `get_response()` is not called in the sequence code for every transaction.

   To resolve this error, you can either add `get_response()` in the sequence for each transaction, or you can simply turnoff this error.

   For example:

   ```
   `uvm_do(req)
   ```

```
        get_response(rsp);
```

Note
To turn off this error, use the UVM method `set_response_queue_error_report_disabled()` in the constructor of the sequence. For example, `set_response_queue_error_report_disabled(1);`

2. You may see this error even though `get_response()` is called for every transaction, if the number of outstanding transactions is high.

   It can happen that more than eight responses can be received by the master at the same point of time, and can cause response queue overflow resulting in the error.

   To resolve the error, you need to increase the depth of the response queue using the UVM method `set_response_queue_depth()` to a large value (as per the outstanding transaction setting).

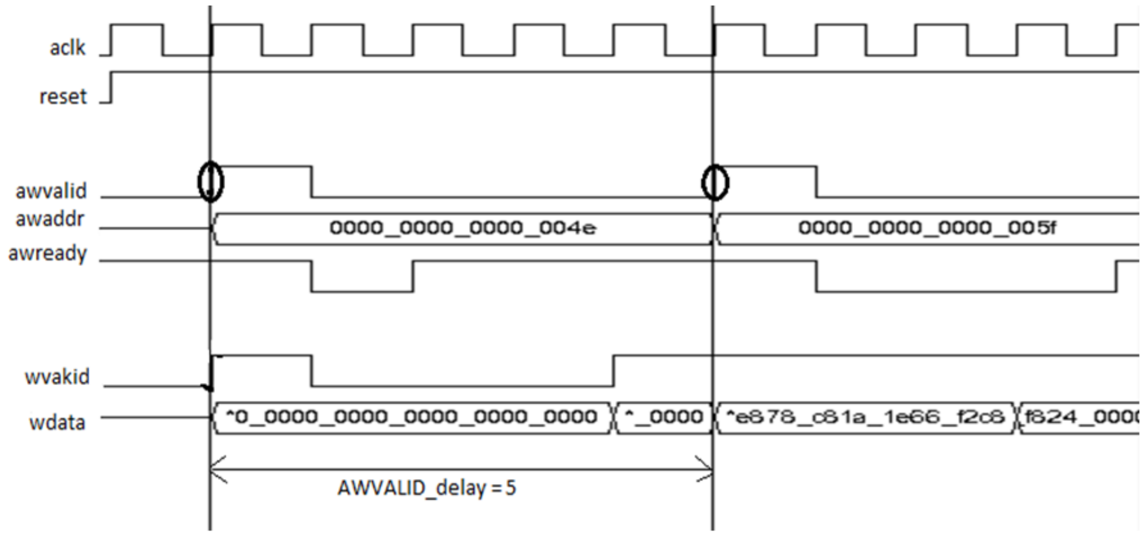
   Call the following method inside the constructor of the sequence.

   For example, `set_response_queue_depth(30);`

1.5 AXI delay control attributes

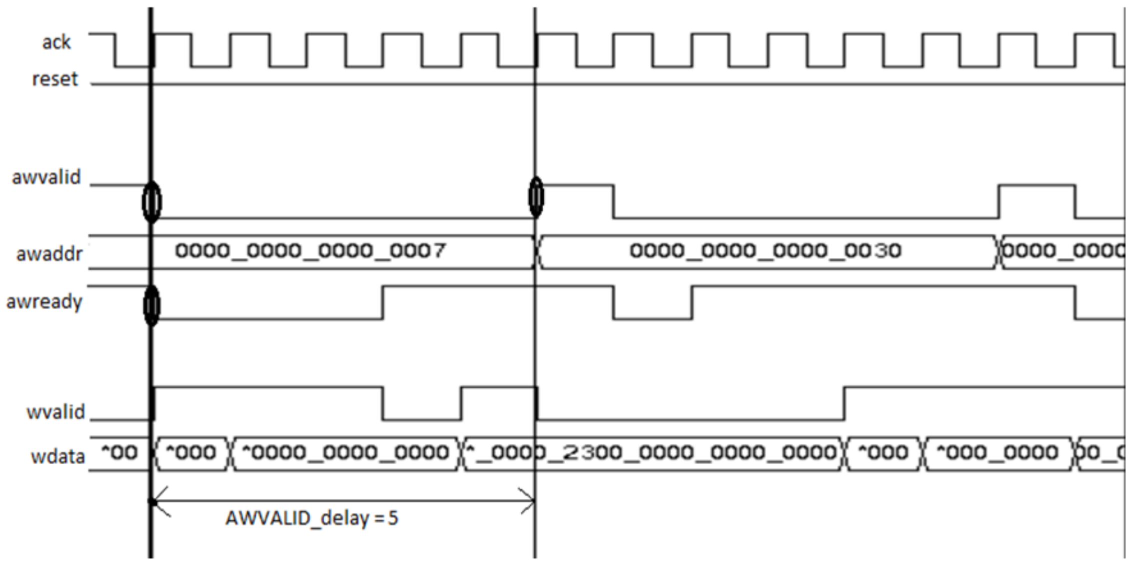
1.5.1 Description of address_valid_delay for reference event PREV_ADDR_VALID for write address channel

Addr_valid_delay defines the number of cycles the AWVALID signal is delayed with respect to the reference event PREV_ADDR_VALID. It is applicable only for the ACTIVE MASTER.



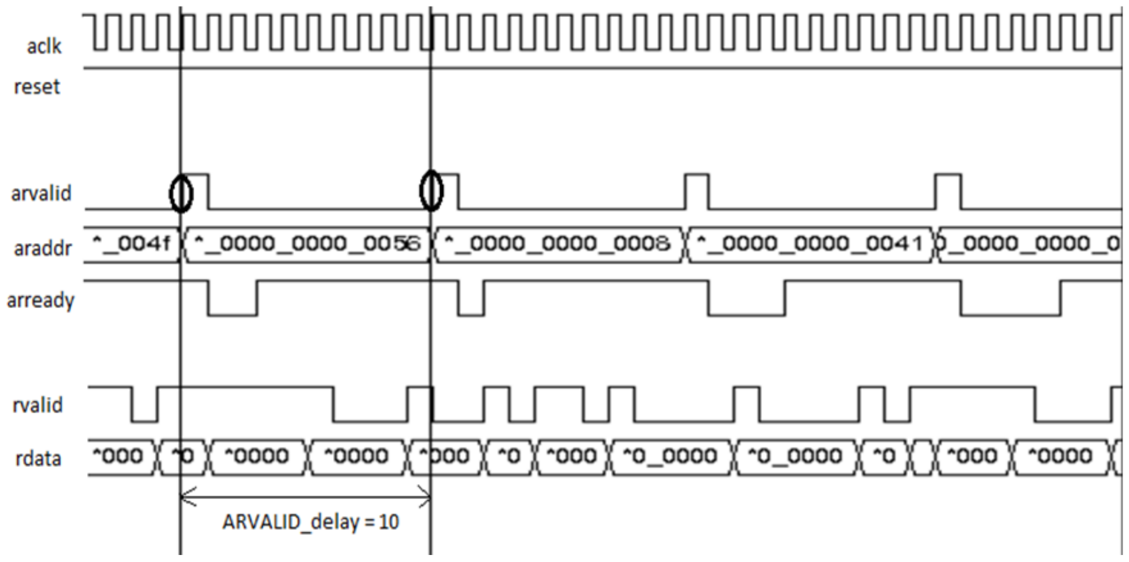
1.5.2 Description of address_valid_delay for reference event PREV_ADDR_HANDSHAKE for write address channel

Addr_valid_delay defines the number of cycles the AWVALID signal is delayed with respect to the reference event PREV_ADDR_HANSHAKE. It is applicable only for the ACTIVE MASTER.



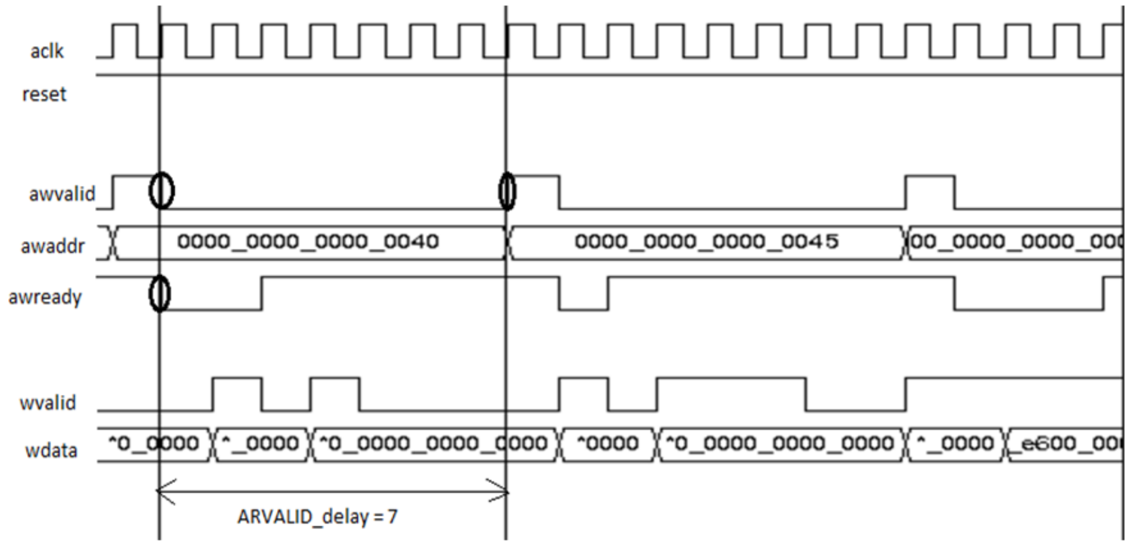
1.5.3 Description of address_valid_delay for reference event PREV_ADDR_VALID for read address channel

Addr_valid_delay defines the number of cycles the ARVALID signal is delayed with respect to the reference event PREV_ADDR_VALID. It is applicable only for the ACTIVE MASTER.



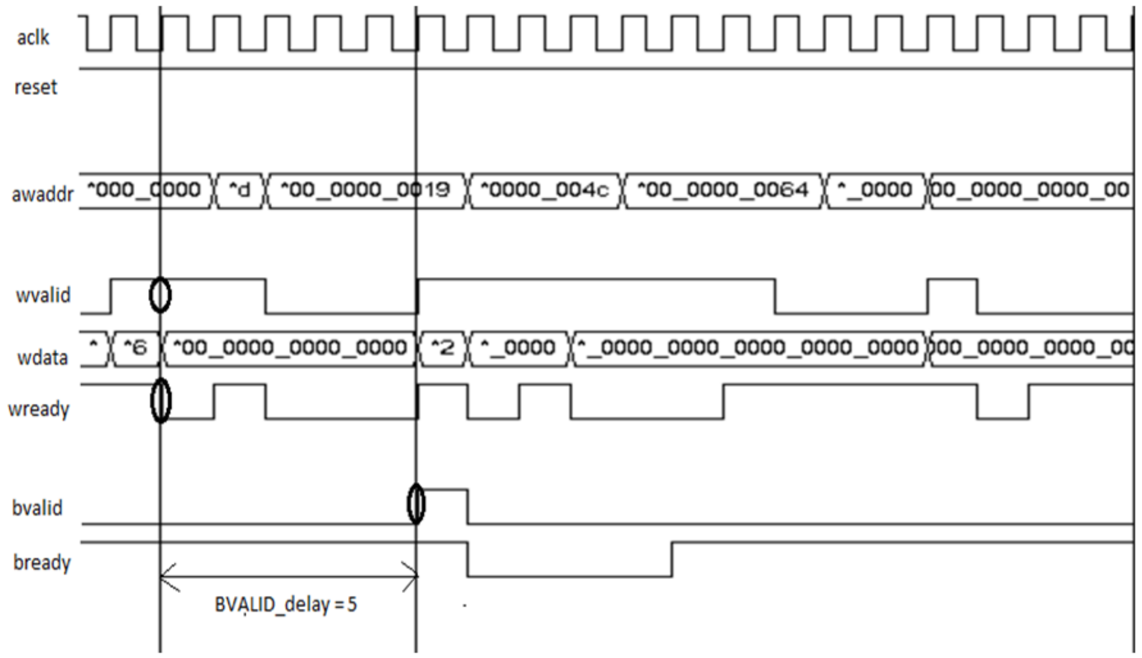
1.5.4 Description of address_valid_delay for reference event PREV_ADDR_HANDSHAKE for read address channel

Addr_valid_delay defines the number of cycles the ARVALID signal is delayed with respect to the reference event PREV_ADDR_HANDSHAKE. It is applicable only for the ACTIVE MASTER.



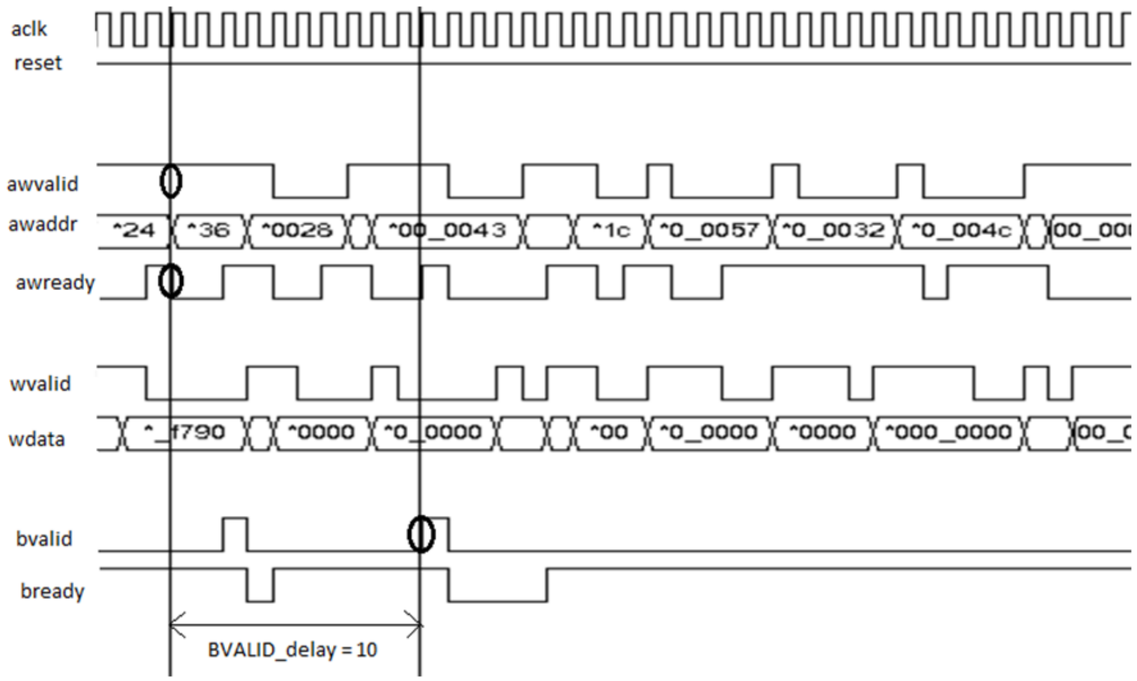
1.5.5 Description of bvalid_delay for reference event LAST_DATA_HANDSHAKE for write response channel

Represents delay in terms of clock cycle for bvalid assertion with respect to the reference event LAST_DATA_HANDHAKE. It is applicable only for the ACTIVE SLAVE.



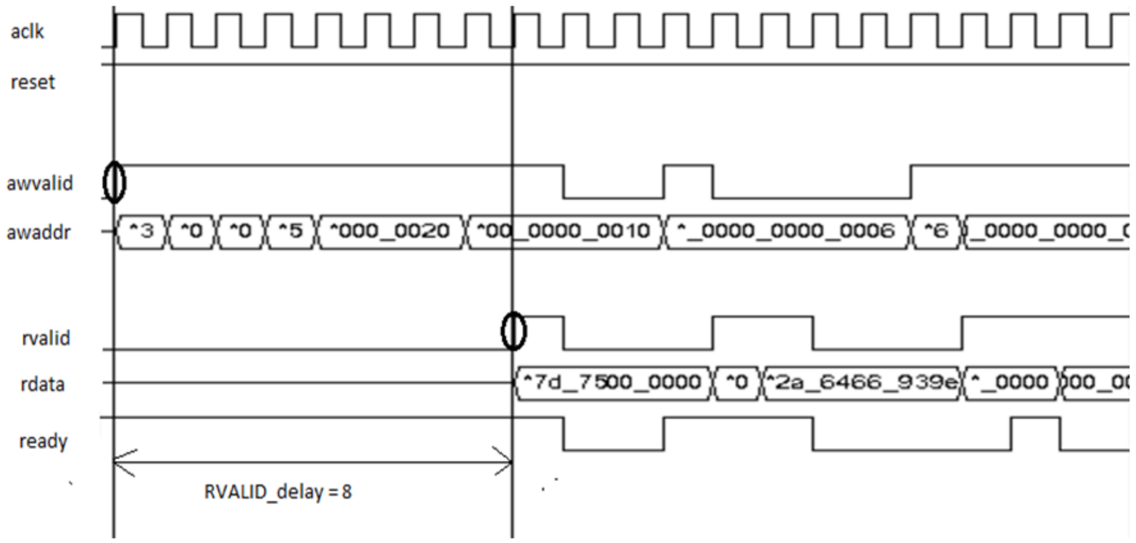
1.5.6 Description of bvalid_delay for reference event ADDR_HANDSHAKE for write response channel

Represents delay in terms of clock cycles for bvalid assertion with respect to the reference event ADDR_HANDHAKE. It is applicable only for the ACTIVE SLAVE.



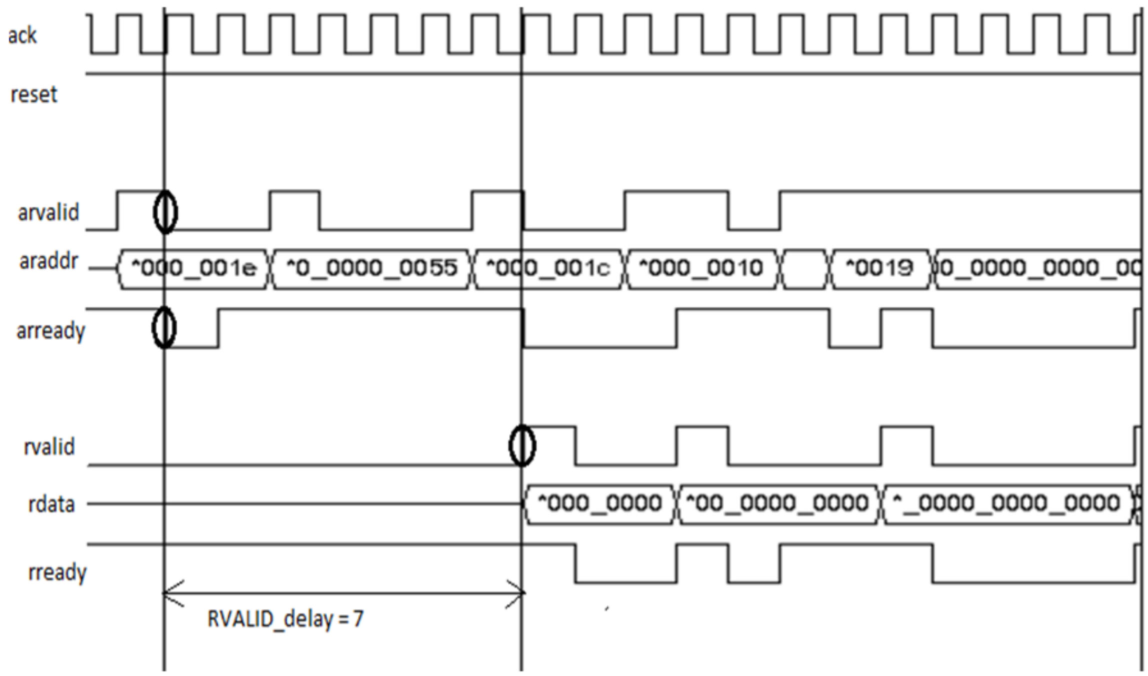
1.5.7 Description of first rvalid_delay for reference event READ_ADDR_VALID for read data channel

Represents a delay for the first RVALID assertion with respect to the reference event READ_ADDR_VALID. It is applicable only for the ACTIVE SLAVE.



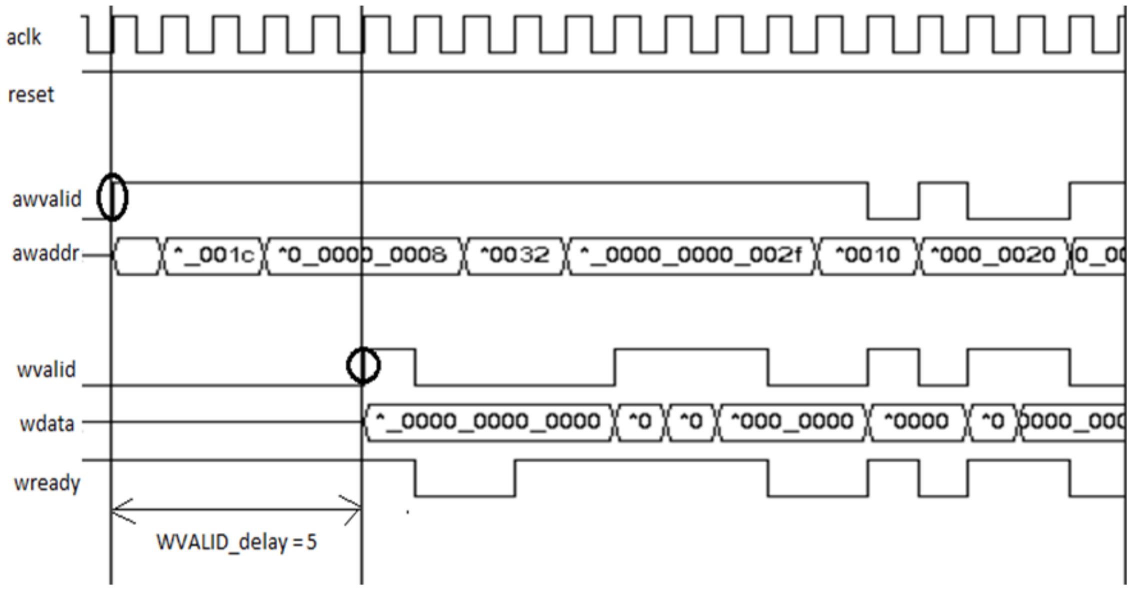
1.5.8 Description of first rvalid_delay for reference event READ_ADDR_HANDSHAKE for read data channel

Represents a delay for the first RVALID assertion with respect to the reference event READ_ADDR_HANDSHAKE.



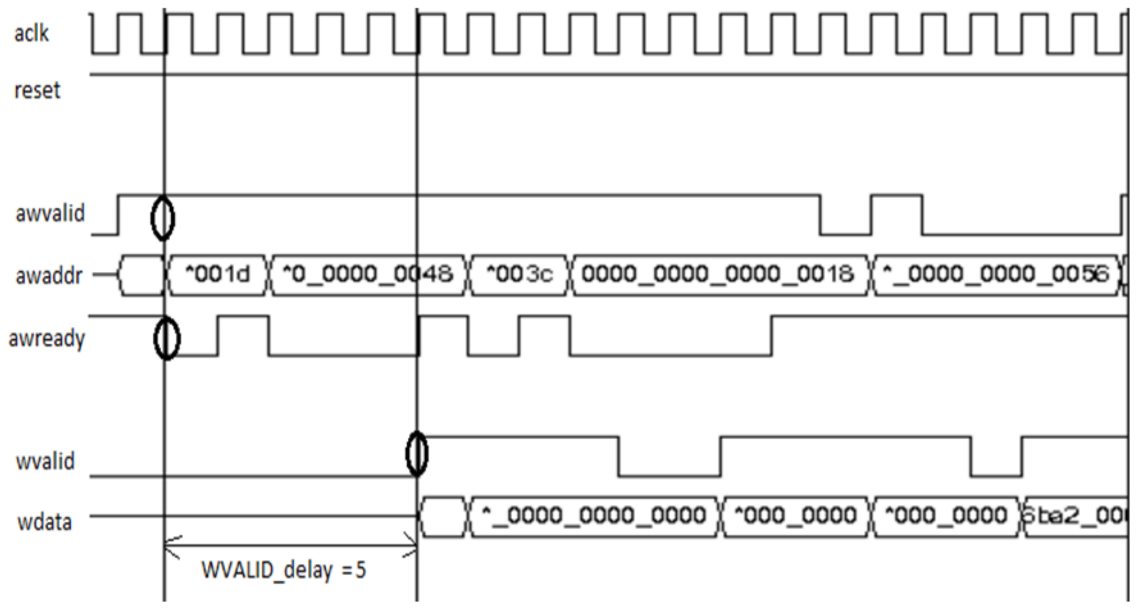
1.5.9 Description of first wvalid_delay for reference event WRITE_ADDR_VALID for write data channel

Represents a delay for the first WVALID assertion with respect to the reference event WRITE_ADDR_VALID.



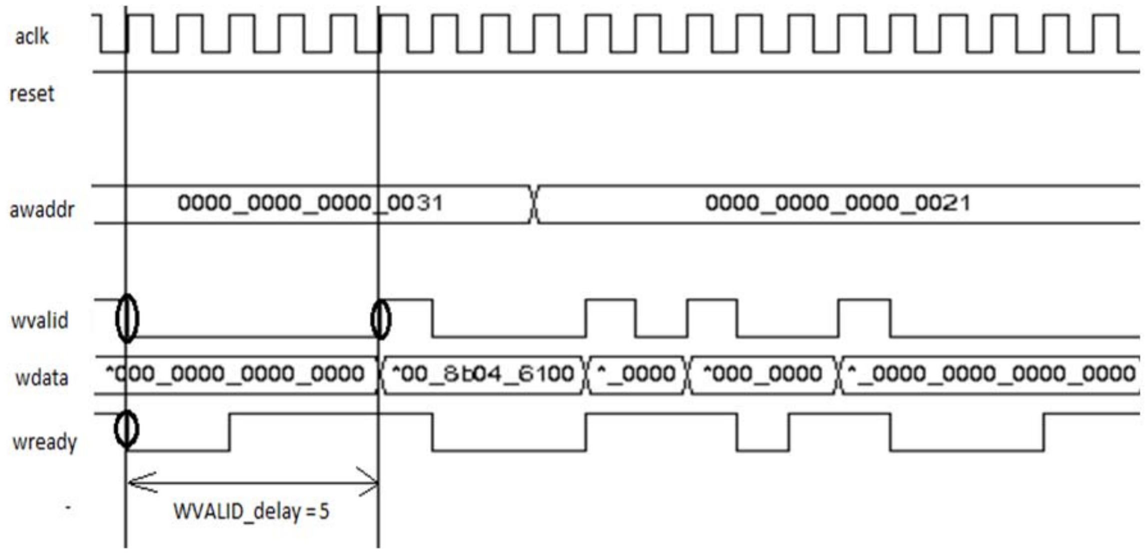
1.5.10 Description of first wvalid_delay for reference event WRITE_ADDR_HANDSHAKE for write data channel

Represents a delay for the first WVALID assertion with respect to the reference event WRITE_ADDR_HANDSHAKE.



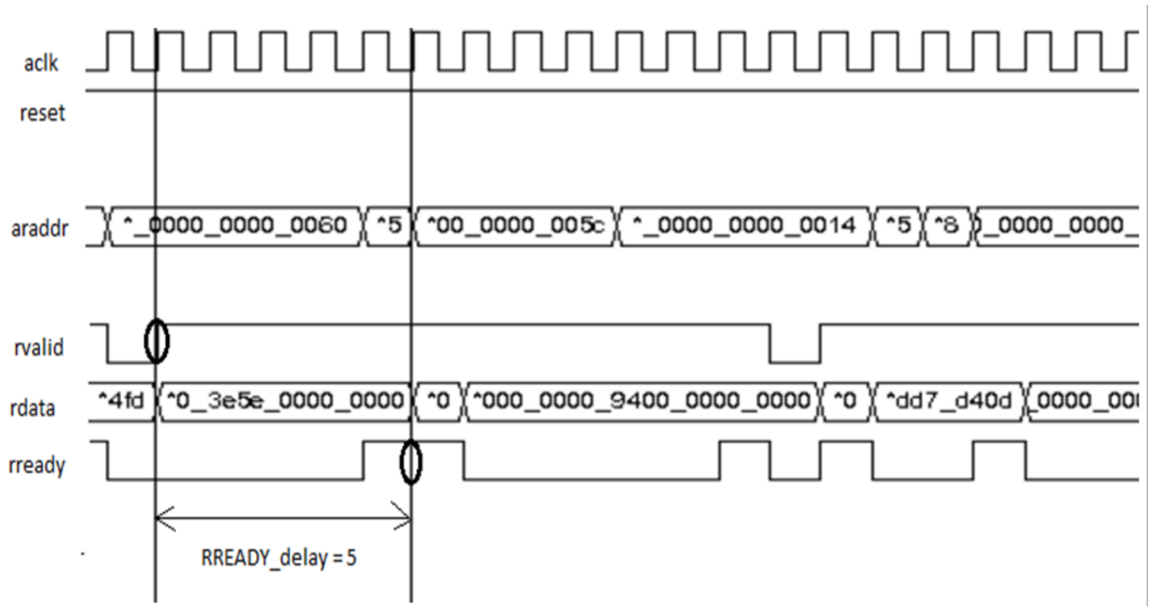
1.5.11 Description of first wvalid_delay for reference event PREV_WRITE_DATA_HANDSHAKE for write data channel

Represents a delay for the first WVALID assertion with respect to the reference event PREV_WRITE_DATA_HANDSHAKE.



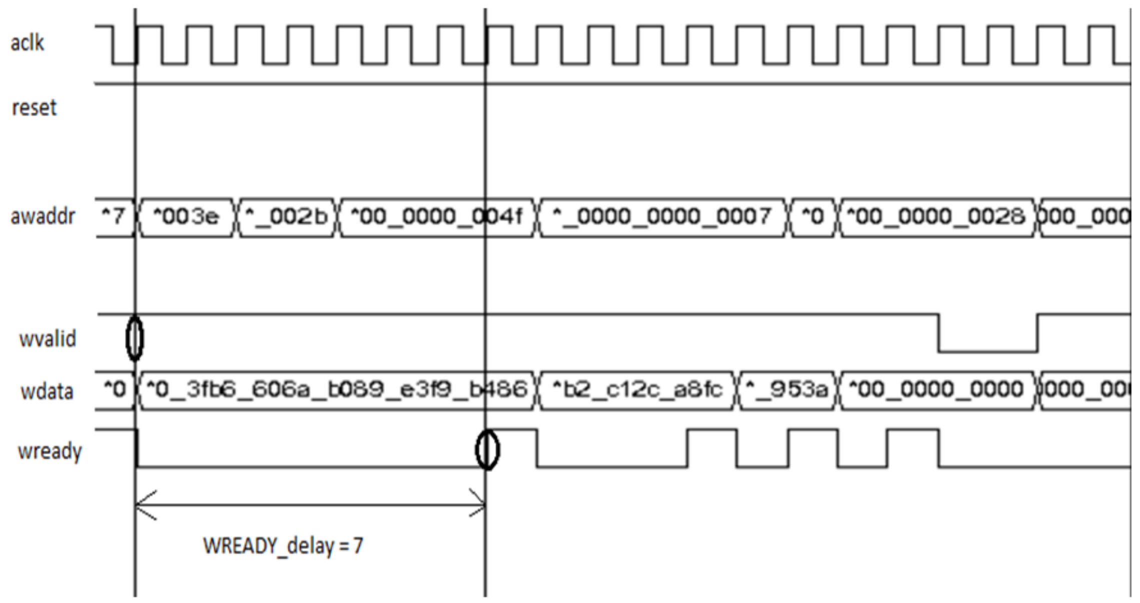
1.5.12 Description of rready_delay for reference event RVALID for read data channel

Represents a delay for the first RREADY assertion with respect to the reference event RVALID.



1.5.13 Description of wready_delay for reference event WVALID for write data channel

Represents a delay for the first WREADY assertion with respect to the reference event WVALID.

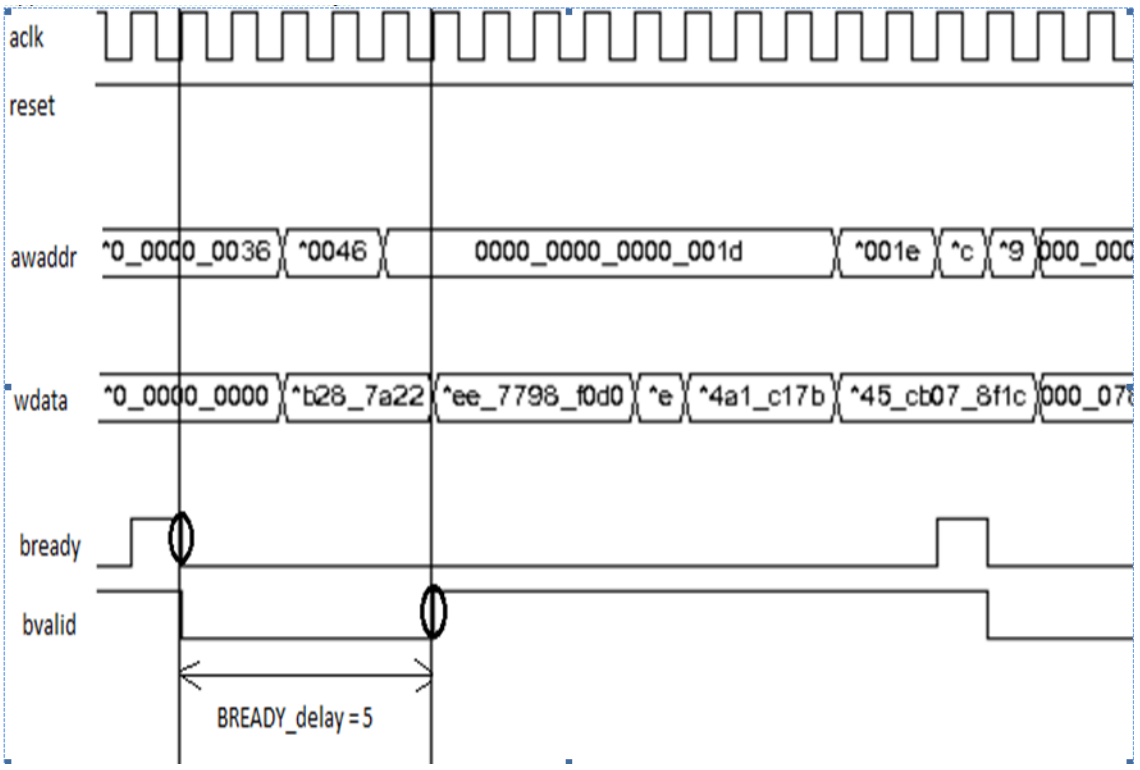


1.5.14 Description of bready_delay for reference event BVALID for write response channel

If configuration parameter `svt_axi_port_configuration` :: `default_bready` is FALSE, this member defines the BREADY signal delay in number of clock cycles. The reference event for this delay is assertion of bvalid.

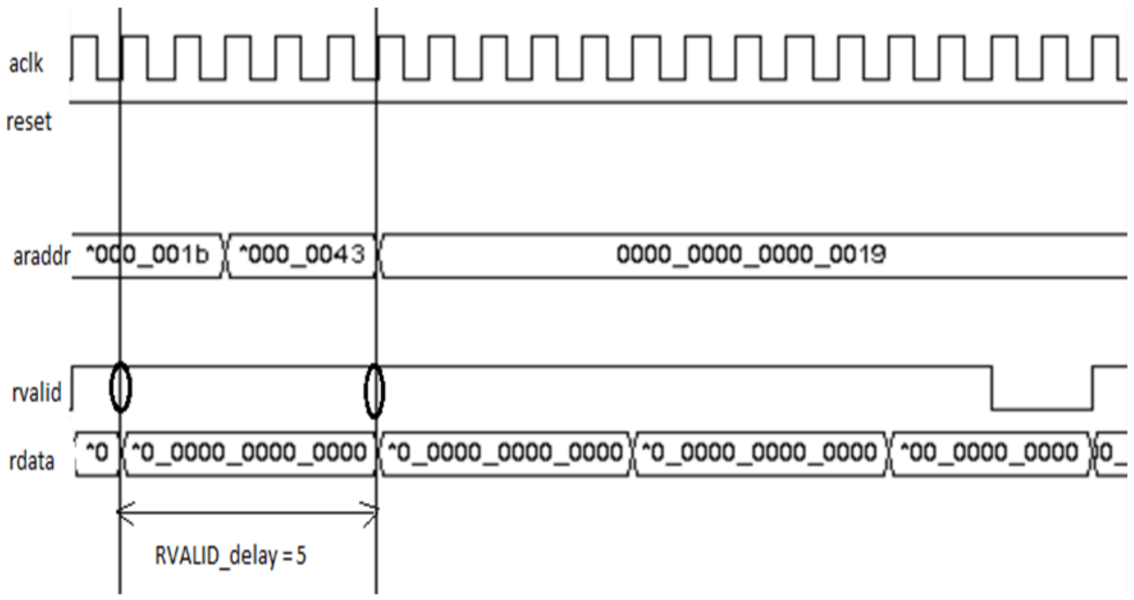
If the configuration parameter `svt_axi_port_configuration` :: `default_bready` is TRUE, this member defines the number of clock cycles for which BREADY signal should be deasserted after each handshake, before pulling it up again to its default value.

Represents the bready delay for reference event BVALID for `default_bready` is TRUE. It is applicable only for the ACTIVE MASTER.



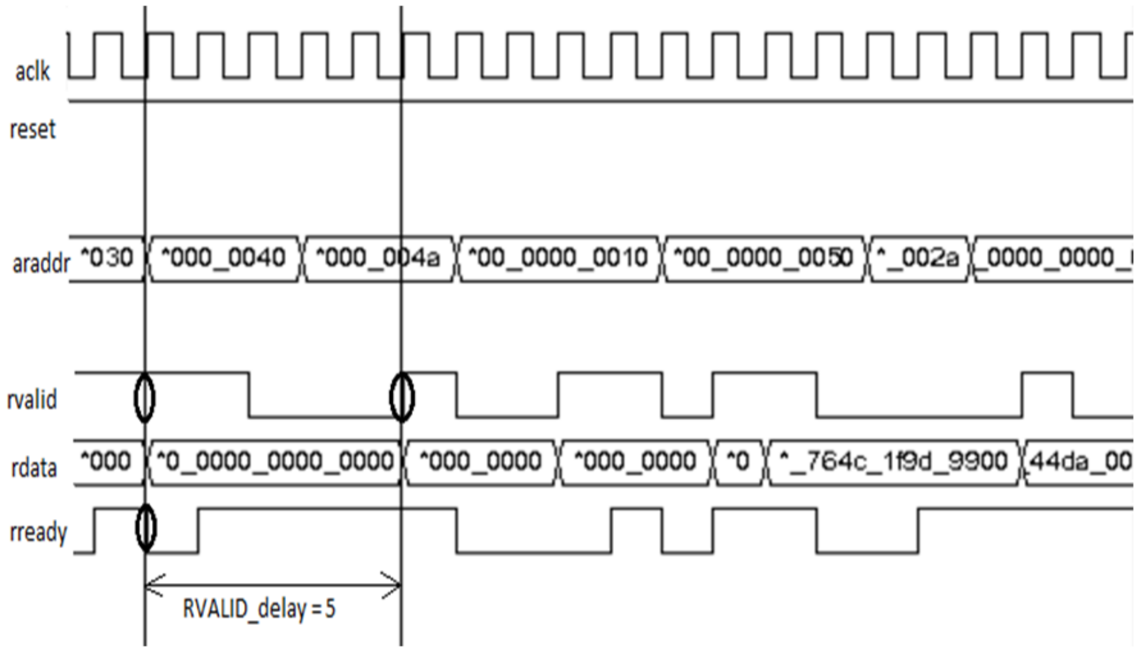
1.5.15 Description of next rvalid_delay for reference event PREV_RVALID for read data channel

Represents a delay for the next RVALID assertion with respect to the reference event PREV_RVALID.



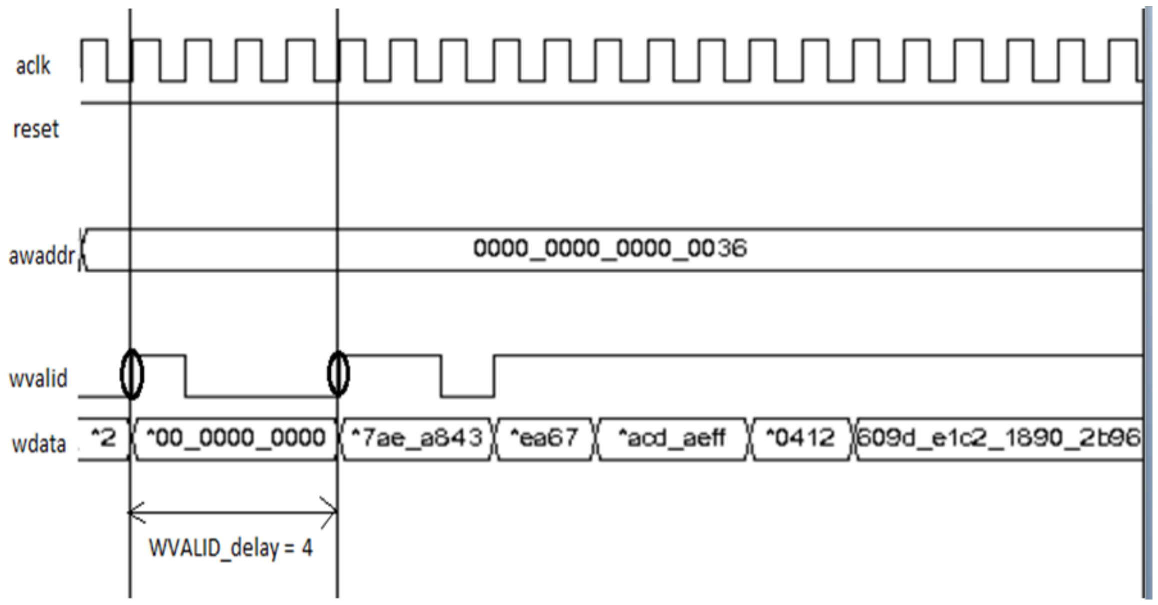
1.5.16 Description of next rvalid_delay for reference event PREV_READ_HANDSHAKE for read data channel

Represents a delay for the next RVALID assertion with respect to the reference event PREV_READ_HANDSHAKE.



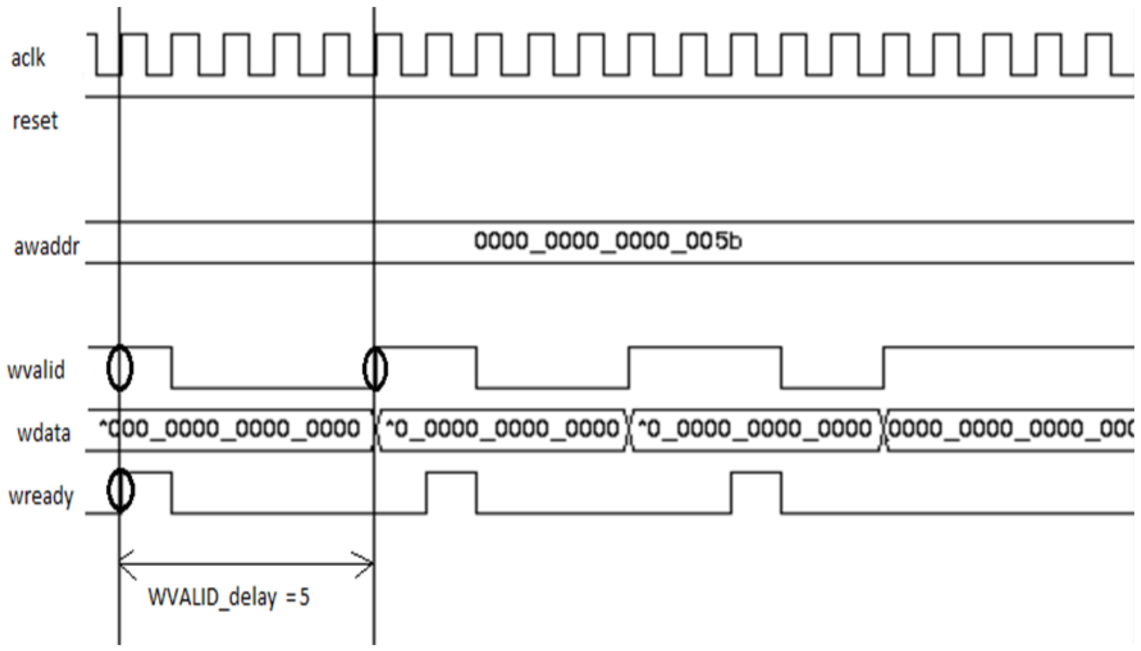
1.5.17 Description of next wvalid_delay for reference event PREV_WVALID for write data channel

Represents a delay for the next WVALID assertion with respect to the reference event PREV_WVALID.



1.5.18 Description of next wvalid_delay for reference event PREV_WRITE_HANDSHAKE for write data channel

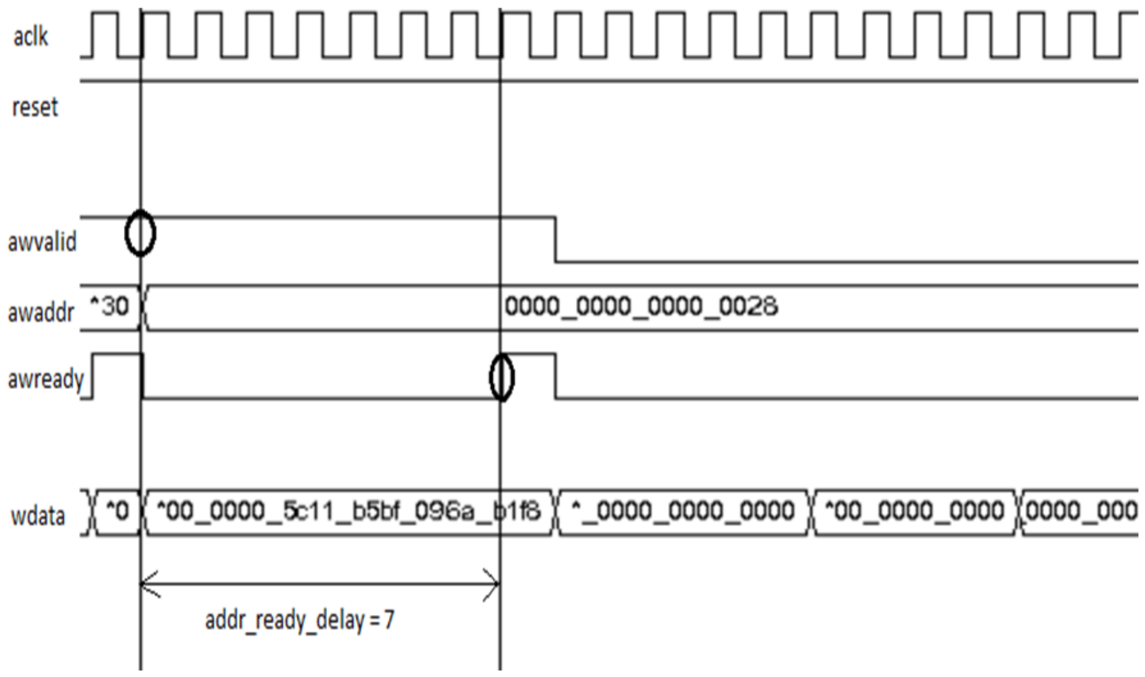
Represents a delay for the next WVALID assertion with respect to the reference event PREV_WRITE_HANDSHAKE.



1.5.19 Description of addr_ready delay for reference event ADDR_VALID for write address channel

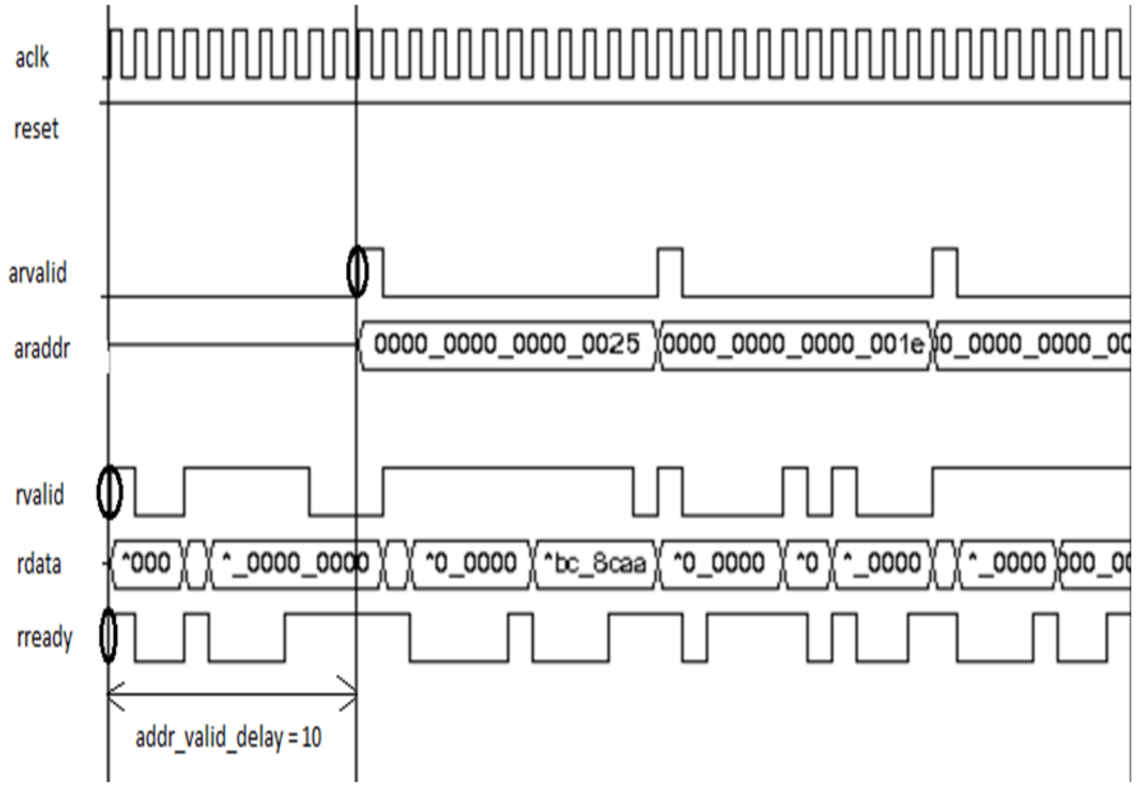
If the configuration parameter `svt_axi_port_configuration :: default_awready` is FALSE, this member defines the AWREADY signal delay in number of clock cycles. The reference event used for this delay is addr_valid.

If the configuration parameter `svt_axi_port_configuration :: default_awready` is TRUE, this member defines the number of clock cycles for which AWREADY signal should be deasserted after each handshake, before pulling it up again to its default value. It is applicable only for the ACTIVE SLAVE.



1.5.20 Description of addr_valid_delay for reference event FIRST_DATA_HANDSHAKE_DATA_BEFORE_ADDR for read address channel

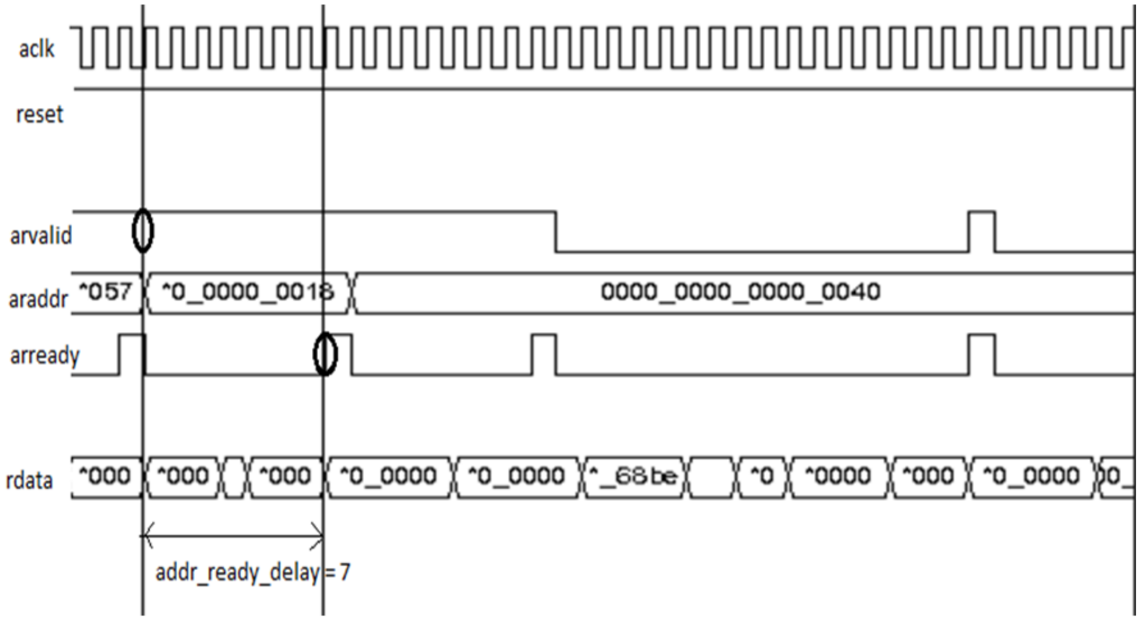
Addr_valid_delay defines the number of cycles the ARVALID signal is delayed with respect to the reference event FIRST_DATA_HANDSHAKE_DATA_BEFORE_ADDR. It is applicable only for the ACTIVE MASTER.



1.5.21 Description of addr_ready_delay for reference event ADDR_VALID for read address channel

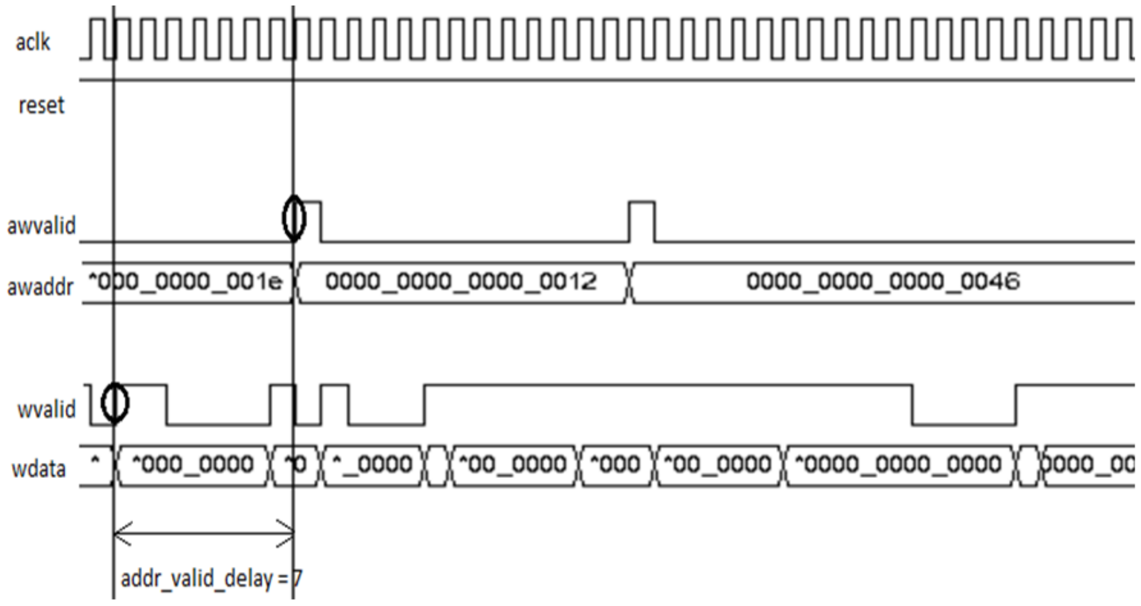
If the configuration parameter `svt_axi_port_configuration :: default_arready` is FALSE, this member defines the ARREADY signal delay in number of clock cycles. The reference event used for this delay is addr_valid.

If the configuration parameter `svt_axi_port_configuration :: default_arready` is TRUE, this member defines the number of clock cycles for which ARREADY signal should be deasserted after each handshake, before pulling it up again to its default value. It is applicable only for the ACTIVE SLAVE.



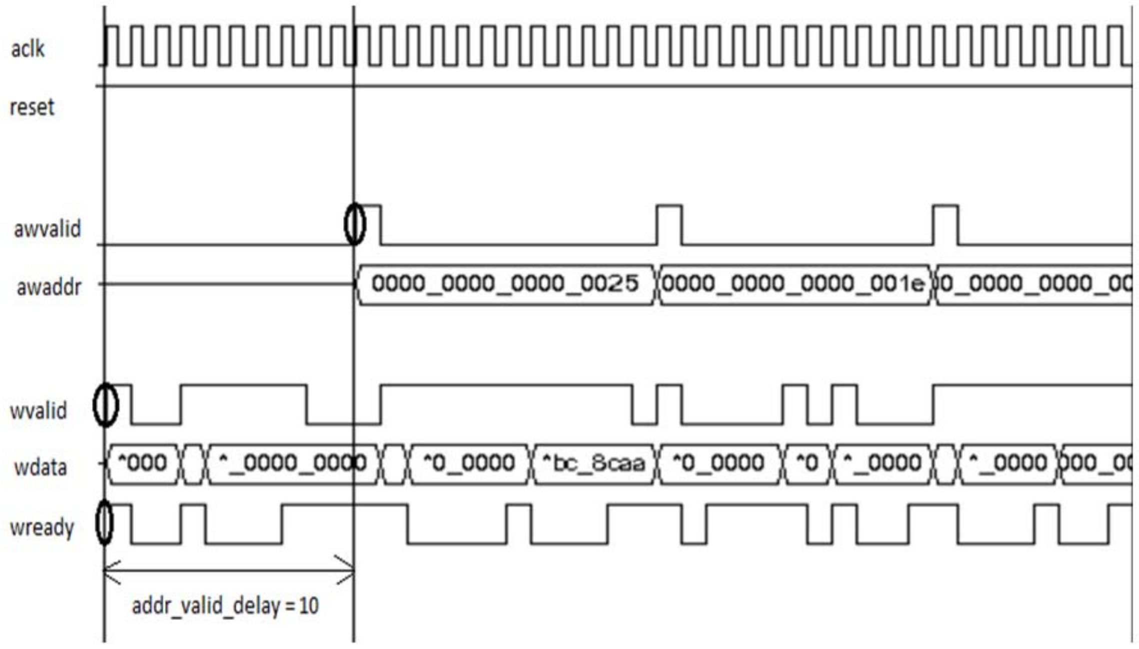
1.5.22 Description of addr_valid_delay for reference event FIRST_WVALID_DATA_BEFORE_ADDR for write address channel

Addr_valid_delay defines the number of cycles the AWVALID signal is delayed with respect to the reference event FIRST_WVALID_DATA_BEFORE_ADDR. It is applicable only for the ACTIVE MASTER.



1.5.23 Description of addr_valid_delay for reference event FIRST_DATA_HANDSHAKE_DATA_BEFORE_ADDR for write address channel

Addr_valid_delay defines the number of cycles the AWVALID signal is delayed with respect to the reference event FIRST_DATA_HANDSHAKE_DATA_BEFORE_ADDR. It is applicable only for the ACTIVE MASTER.



1.6 Related SolvNet Articles

Doc Id	Methodology Tool	AXI
038043	ALL	AXI-SVT: Write Data and Write Strobe Alignment in Transactions
038063	ALL	AXI-SVT: How to Disable Default Constraints for Delays as Specified in AXI VIP Master Transaction Class
036238	ALL	SVT-VIP: How do I view bus transactions at a high-level
1492298	ALL+VCS	SVT-AXI: How to generate code coverage
037044	General	AXI-SVT: Why All the WSTRB Bits are Not Set to '1'
032748	General	SVT-AXI: How to enable configuration aware coverage
019013	General	SVT-AXI: How to turn on coverage and report generation using the AXI Monitor
037699	General	SVT-AXI: How to connect master/slave VIP interface pins with slave/master DUT pins
040413	General + VMM	AXI-SVT: Leveraging VIP Provided MSSG in axi_system_group to Send Controlled Traffic from AXI Master
036097	UVM	AXI-SVT:C reating a New UVM AXI Sequence With Discovery VIP

Doc Id	Methodology Tool	AXI
033528	UVM	AXI-SVT: How to Override Slave Sequence Item Type for AXI SVT UVM VIP
038051	UVM	AXI-SVT: Example of Error Injection Scenario
041145	UVM	AXI-SVT: Using Transaction id_width of Size Greater Than 8
040534	UVM	AXI-SVT: Specifying a Zero Delay Between AWVALID and AWREADY While Using the Interconnect VIP
040339	UVM	AXI-SVT: Data Before Address Transaction
040341	UVM	AXI-SVT: Defining the Initialization Value for AXI VIP Ports
040752	UVM	AXI-SVT: Specifying a Master Sequence Library to Run on an AXI MasterSequencer and Populate it with User-Defined Master Sequence
040961	UVM	AXI-SVT: Why in Some Cases AWREADY is Not Coming From Slave DUT
035833	UVM	AXI-SVT: Example for Usage of Passive Agents in UVM
035976	UVM	AXI-SVT: Example for Multiple axi_system_env Usage
034476	UVM	AXI-SVT: An Example of Multi Master and Multi Slave Connected Through Interconnect and Slave With Memory Preload
037681	UVM	AXI-SVT: How to Use System Monitor Across Interconnect Modifying Address?
033572	UVM	AXI-SVT: How to Override bus_inactivity_timeout Warning Message?
037046	UVM	AXI-SVT: How to Use Stand-Alone Master and Slave Agents in UVM?
037333	UVM	AXI-SVT: How to Enable Separate Number of Outstanding Transactions for Read/Write Channels in UVM Env?
037612	UVM	AXI-SVT: Master Sequencer Error for "Response Queue Overflow, Response was Dropped"
037846	UVM	AXI-SVT: How to Initiate Back-to-Back Write Transactions from Master?
039040	UVM	AXI-SVT: How to Use External User-Defined UVM Sequencer?
038988	UVM	AXI-SVT: Back to Back Transfers
039289	UVM	AXI-SVT: Tracking AXI Transaction for its Protocol Phase Completion
036978	UVM	AXI-SVT: Adding a User-Defined Checker to AXI UVM VIP
036640	UVM	AXI-SVT: Defining the Coverage Test Name Using Code
039243	UVM	AXI-SVT: Increasing the bready Delay to a Value Greater than 16
039313	UVM	AXI-SVT: Specifying delay between two beats for write beats
039743	UVM	AXI-SVT: Randomization of wready Signal from AXI Slave
039204	UVM	AXI-SVT: Randomly De-Asserting BREADY Signal in AXI UVM VIP
034030	UVM	How to Use AXI SVT VIP to Work With DUT Which Supports Only Write Channel?
035634	UVM	AXI-SVT: Handling Delayed Response From Slave VIP Model

Doc Id	Methodology Tool	AXI
040575	UVM	AXI-SVT: How to Get a Count of Outstanding Transactions With AXI SVT VIPs Slave Component?
1534657	UVM	SVT-AXI: port monitor error "register_fail:awsize_data_width_active_check"
1532540	UVM	SVT-AXI: Master Sequencer Error for "Response Queue Overflow, Response was Dropped".
1512706	UVM	SVT-AXI: Accessing FIFO memory for INCR bursts of burst length equal to 1
1745602	UVM	SVT-AXI: How to get the data of each beat of AXI VIP transaction as it progresses?
039038	UVM	SVT-AXI: How to build ACE tests for system-level with VIP built-in sequences?
036153	UVM	SVT-AXI: How to Access ACE Master VIP Built-In Cache Model in Sequence?
036156	UVM	SVT-AXI: How to generate user-defined ACE snoop response?
038309	UVM	SVT-VIP: How to leverage standalone AXI and AHB system configurations used in axi/ahb_system_env to amba_system_env system configuration?
033253	UVM	SVT-AXI: How to disable a particular protocol check?
1734047	UVM	SVT-AXI: How to send a transaction object from a callback function to scoreboard?
1726752	UVM	SVT-AXI: How to get a transaction object at the end of write data phase, and before the write response phase for a write transaction?
1561135	UVM	SVT-AXI: How to use master sequence library with virtual sequence and configure the sequence library?
1624755	UVM	SVT-AXI: How to generate exclusive access from master?
038809	UVM	SVT-AXI: How to specify different delay weights for master or slave transactions?
1625071	UVM	SVT-AXI: How to generate locked access from master VIP?
038040	UVM	SVT-AXI: How to override the default delays for meta delays?
1811603	UVM	SVT-AXI: How to add a user defined covergroup?
1800368	UVM	SVT-AXI: How to generate write responses out of order?
035634	UVM	SVT-AXI: How to handle delayed response from slave VIP Model?
037045	UVM + RAL	AXI SVT: How to Initiate Register Read/Write Sequence Alongside AXI Data Sequence?
2070890	UVM + RAL	AXI-SVT: UVM RAL Example
040442	UVM + TLM	AXI-SVT: Extending AXI SLAVE VIP for TLM Socket
037331	UVM+TLM	SVT-AXI: How to add user-defined TLM analysis ports into port monitor callbacks?
1562967	UVM+VCS	SVT-AXI: How to use VIPs with Verdi, creating dump in fsdb format?
037915	VHDL	AXI-SVT: Working VIP Example with VHDL DUT

Doc Id	Methodology Tool	AXI
023357	VMM	AXI-SVT: White Paper on Using Synopsys AMBA AXI VMM VIP
034745	VMM	Too Many Arguments to Function/Task Call With AXI SVT VMM VIP
036467	VMM	AXI-SVT: Example for Standalone Master in VMM
036479	VMM	AXI-SVT: Example of AXI VIP (VMM) in Explicit Environment (vmm_env)
036575	VMM	AXI-SVT: How to Use AXI VIP in Explicitly Phased vmm_subenv
037334	VMM	AXI-SVT: How to Enable Separate Number of Outstanding Transactions for Read/Write Channels in VMM Env?
037332	VMM	AXI-SVT: How do I add user defined TLM analysis ports into Port Monitor callbacks in a VMM environment?
032111	VMM	SVT-AXI: Disable specific covergroup from AXI default coverage class
038507	VMM	SVT-AXI: How to track the delivery of transactions captured by AXI monitor to scoreboard?