Stream
Model Independent Transaction
User Guide

User Guide for Release 2020.07

By

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

http://www.SynthWorks.com

Table of Contents

1 Overview

The Stream Model Independent Transaction package (StreamTransactionTransactionPkg.vhd) defines a record for communication and transaction initiation procedures that are suitable for Stream Interfaces.

2 Stream Transaction Record

The Address Bus Transaction Record (StreamRecType) defines the transaction interface between the test sequencer and the verification component. As such, it is the primary channel for information exchange between the two. It is defined as follows.

```
type StreamRecType is record
  -- Handshaking controls
  --   Used by RequestTransaction in the Transaction Procedures
  --   Used by WaitForTransaction in the Verification Component
  --   RequestTransaction and WaitForTransaction are in osvvm.TbUtilPkg
  Rdy             : bit_max ;
  Ack             : bit_max ;
  -- Transaction Type
  Operation       : StreamOperationType ;
  -- Data and Transaction Parameter to and from verification component
  DataToModel     : std_logic_vector_max_c ;
  ParamToModel    : std_logic_vector_max_c ;
  DataFromModel   : std_logic_vector_max_c ;
  ParamFromModel  : std_logic_vector_max_c ;
  -- Verification Component Options Parameters - used by SetModelOptions
  IntToModel      : integer_max ;
  BoolToModel     : boolean_max ;
  IntFromModel    : integer_max ;
  BoolFromModel   : boolean_max ;
  TimeToModel     : time_max ;
  TimeFromModel   : time_max ;
  -- Verification Component Options Type - currently aliased to type integer_max
  Options         : integer_max ;
end record StreamRecType ;
```

The record element types, bit_max, std_logic_vector_max_c, integer_max, time_max, and boolean_max are defined in the OSVVM package ResolutionPkg. These types allow the record to support multiple drivers and use resolution functions based on function maximum (return largest value).

3 Usage of the Record Interface

The data and parameter fields of the record are unconstrained. Unconstrained objects may be used on component/entity interfaces. The record will be sized when used as a record signal in the test harness of the testbench. Such a declaration is shown below.

```
signal AxiStreamTransmitterTransRec : StreamRecType(
    DataToModel  (AXI_DATA_WIDTH-1 downto 0),
    DataFromModel(AXI_DATA_WIDTH-1 downto 0),
    ParamToModel  (0 downto 1),                  -- Not Used for AXI Stream
    ParamFromModel(0 downto 1)                   -- Not Used for AXI Stream
```

```
) ;
```

4 Transaction types

Transactions can be either directive or interface transactions. A directive transaction interacts with the model without interacting with the DUT interface.

Interface transactions can be blocking or non-blocking. Interface transactions without any additional naming are blocking transactions. Blocking transactions do not complete until the transaction completes.

Non-blocking interface transactions return immediately. Non-blocking transactions are denoted as either Asynchronous or Try. An asynchronous transaction provides transaction information but does not wait for it to complete. A Try transaction queries if transaction information is available, such as read data, and if it is returns it.

5 Directive Transactions

Directive transactions interact with the verification component without generating any transactions or interface waveforms. These transactions are supported by all verification components.

```
------------------------------------------------------------
procedure WaitForTransaction (
--  Wait until pending (transmit) or next (receive) transaction(s) complete
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType
) ;

------------------------------------------------------------
procedure WaitForClock (
-- Wait for NumberOfClocks number of clocks
-- relative to the verification component clock
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  constant  WaitCycles      : in    natural := 1
) ;

------------------------------------------------------------
procedure GetAlertLogID (
-- Get the AlertLogID from the verification component.
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  variable  AlertLogID      : out   AlertLogIDType
) ;

------------------------------------------------------------
procedure GetErrorCount (
-- Error reporting for testbenches that do not use OSVVM AlertLogPkg
-- Returns error count.  If an error count /= 0, also print errors
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  variable  ErrorCount      : out   natural
) ;
```

```
------------------------------------------------------------
procedure GetTransactionCount (
-- Get the number of transactions handled by the model.
------------------------------------------------------------
  signal    TransactionRec   : inout StreamRecType ;
  variable  TransactionCount : out   integer
) ;
```

6 Model Options

Model operations are directive transactions that are used to configure the verification component. They can either be used directly or with a model specific wrapper around them - see AXI models for examples.

```
------------------------------------------------------------
procedure SetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  constant OptVal      : In    boolean
) ;

------------------------------------------------------------
procedure SetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  constant OptVal      : In    integer
) ;

------------------------------------------------------------
procedure SetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  constant OptVal      : In    std_logic_vector
) ;

------------------------------------------------------------
procedure SetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  constant OptVal      : In    time
) ;

------------------------------------------------------------
procedure GetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  variable OptVal      : Out   boolean
) ;
```

```
------------------------------------------------------------
procedure GetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  variable OptVal      : Out   integer
) ;

------------------------------------------------------------
procedure GetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  variable OptVal      : Out   std_logic_vector
) ;

------------------------------------------------------------
procedure GetModelOptions (
------------------------------------------------------------
  signal   TransRec    : InOut StreamRecType ;
  constant Option      : In    integer ;
  variable OptVal      : Out   time
) ;
```

7 Transmitter Transactions

```
------------------------------------------------------------
procedure Send (
-- Blocking Send Transaction.
-- Param is an extra parameter to be used by the verification component
-- The UART verification component uses Param for error injection.
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  constant  Param           : in    std_logic_vector ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;

procedure Send (
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;

------------------------------------------------------------
procedure SendAsync (
-- Asynchronous / Non-Blocking Send Transaction
-- Param is an extra parameter to be used by the verification component
-- The UART verification component uses Param for error injection.
------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
```

```
    constant  Param           : in    std_logic_vector ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;

  procedure SendAsync (
    signal    TransactionRec  : inout StreamRecType ;
    constant  Data            : in    std_logic_vector ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;
```

8 Receiver Transactions

```
  ------------------------------------------------------------
  procedure Get (
  -- Blocking Get Transaction.
  -- Param is an extra parameter used by the verification component
  -- The UART verification component uses Param for error injection.
  ------------------------------------------------------------
    signal    TransactionRec  : inout StreamRecType ;
    variable  Data            : out   std_logic_vector ;
    variable  Param           : out   std_logic_vector ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;

  procedure Get (
    signal    TransactionRec  : inout StreamRecType ;
    variable  Data            : out   std_logic_vector ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;

  ------------------------------------------------------------
  procedure TryGet (
  -- Try Get Transaction
  -- If Data is available, get it and return available TRUE,
  -- otherwise Return Available FALSE.
  -- Param is an extra parameter used by the verification component
  -- The UART verification component uses Param for error injection.
  ------------------------------------------------------------
    signal    TransactionRec  : inout StreamRecType ;
    variable  Data            : out   std_logic_vector ;
    variable  Available       : out   boolean ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;

  procedure TryGet (
    signal    TransactionRec  : inout StreamRecType ;
    variable  Data            : out   std_logic_vector ;
    variable  Param           : out   std_logic_vector ;
    variable  Available       : out   boolean ;
    constant  StatusMsgOn     : in    boolean := FALSE
  ) ;

  ------------------------------------------------------------
```

```
procedure Check (
-- Blocking Get Transaction.
-- Data is the expected value to be received.
-- Param is an extra parameter used by the verification component
-- The UART verification component uses Param for error injection.
-------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  constant  Param           : in    std_logic_vector ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;

procedure Check (
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;


-------------------------------------------------------------
procedure TryCheck (
-- Try Check Transaction
-- If Data is available, check it and return available TRUE,
-- otherwise Return Available FALSE.
-- Param is an extra parameter used by the verification component
-- The UART verification component uses Param for error injection.
-------------------------------------------------------------
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  constant  Param           : in    std_logic_vector ;
  variable  Available       : out   boolean ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;

procedure TryCheck (
  signal    TransactionRec  : inout StreamRecType ;
  constant  Data            : in    std_logic_vector ;
  variable  Available       : out   boolean ;
  constant  StatusMsgOn     : in    boolean := FALSE
) ;
```

9 Verification Component Support Functions

Verification component support functions help decode the operation value (StreamOperationType) to determine properties about the operation.

```
-------------------------------------------------------------
function IsTry (
-- True when this transaction is an asynchronous or try transaction.
-------------------------------------------------------------
  constant Operation     : in StreamOperationType
) return boolean ;
```

```
------------------------------------------------------------
function IsCheck (
-- True when this transaction is a check transaction.
------------------------------------------------------------
  constant Operation     : in StreamOperationType
) return boolean ;
```

10 About the OSVVM Model Independent Transactions

OSVVM Model Independent Transactions were developed and are maintained by Jim Lewis of SynthWorks VHDL Training. These evolved from methodology and packages developed for SynthWorks' VHDL Testbenches and verification class. They are part of the Open Source VHDL Verification Methodology (OSVVM) model library (osvvm_common), which brings leading edge verification techniques to the VHDL community.

Please support OSVVM by purchasing your VHDL training from SynthWorks.

11 About the Author - Jim Lewis

Jim Lewis, the founder of SynthWorks, has thirty plus years of design, teaching, and problem solving experience. In addition to working as a Principal Trainer for SynthWorks, Mr Lewis has done ASIC and FPGA design, custom model development, and consulting.

Mr. Lewis is chair of the IEEE 1076 VHDL Working Group (VASG) and is the primary developer of the Open Source VHDL Verification Methodology (OSVVM.org) packages. Neither of these activities generate revenue. Please support our volunteer efforts by buying your VHDL training from SynthWorks.

If you find bugs these packages or would like to request enhancements, you can reach me at jim@synthworks.com.

12 References

[1] Jim Lewis, VHDL Testbenches and Verification, student manual for SynthWorks' class.