SysWip

*Alternative Verification*

# AXI4-Stream Interface Verification IP User Manual

**Release 1.0**

## Table of Contents

1.Introduction ..... 3
    Package Hierarchy ..... 3
    Features ..... 3
    Limitations ..... 3
2.AXI4-Stream Master ..... 3
    Commands ..... 4
    Commands Description ..... 4
    Integration and Usage ..... 6
3.AXI4-Stream Slave ..... 7
    Commands ..... 7
    Commands Description ..... 8
    Integration and Usage ..... 10
4.Important Tips ..... 10
    Master Tips ..... 10
    Slave Tips ..... 10

# 1. Introduction

The AXI4-Stream Verification IP described in this document is a solution for verification of AXI4-Stream master and slave devices. The provided AXI4- Stream verification package includes master and slave verification IPs, bus monitor and examples. It will help engineers to quickly create verification environment and test their AXI4-Stream protocol.

## Package Hierarchy

After downloading and unpacking package you will have the following folder hierarchy:

- axi4_stream_vip
  - docs
  - examples
    - sim
    - testbench
  - verification_ip

The Verification IP is located in the *verification_ip* folder. Just copy the content of this folder to somewhere in your verification environment.

## Features

- Easy integration and usage
- Free SystemVerilog source code
- Compliant to AXI4-Stream Protocol
- Operates as a Master or Slave
- Supports 1, 2, 4, 8, 16 and 32 bytes data block size
- Supports up to 8 user bits per byte
- Programmable response type
- Supports wait states injection
- Supports full random timings
- Protocol monitor

## Limitations

- Doesn't support packets with position bytes

# 2. AXI4-Stream Master

The AXI4-Stream Master Verification IP (VIP) initiates transfers on the bus.

## Commands

Commands marked as a *queued* will be put in the one queue and executed sequentially.

Commands marked as a *blocking* will block other commands execution until finishing its own process.

- **Configuration Commands**
    - **setRandDelay(): -** *non queued, non blocking*
    - **setTimeOut(): -** *non queued, non blocking*
- **Data Transfer Commands**
    - **sendData(): -** *queued, non blocking*
    - **genSingleTransfer(): -** *queued, non blocking*
    - **createUserBuf**(): - *non queued, non blocking*
    - **busIdle():** - *queued, non blocking*
- **Other Commands**
    - **startEnv(): -** *non queued, non blocking*
    - **waitCommandDone():-** *queued, blocking*
    - **printStatus():-** *non queued, non blocking*

## Commands Description

All commands are *AXI4STR_m_env* class methods.

- **setRandDelay()**
    - **Syntax**
        - *setRandDelay(minBurst, maxBurst, minWait, maxWait)*
    - **Arguments**
        - *minBurst:* An *int* variable which specifies minimum value for burst length
        - *maxBurst:* An *int* variable which specifies maximum value for burst length
        - *minWait:* An *int* variable which specifies the minimum value for wait cycles
        - *maxWait:* An *int* variable which specifies the maximum value for wait cycles
    - **Description**
        - Enables/Disables bus random timings. To disable random timing all arguments should be set to zero.
- **setTimeOut()**
    - **Syntax**
        - *setTimeOut(readyTimeOut)*
    - **Arguments**

    - *readyTimeOut:* An *int* variable which specifies the maximum wait clock cycles for slave response.
  - **Description**
    - Sets the maximum clock cycles for slave response. If slave delays response the error message will be generated.
- **sendData()**
  - **Syntax**
    - *sendData(inbuff, setLast, tid, tdest)*
  - **Arguments**
    - *inBuff:* 8 bit vector dynamic array that contains data buffer which should be transferred
    - *setLast:* An *int* variable that specifies the value of the *tlast* signal. If set to 1 the *tlast* signal will be set at the last transfer.
    - *tid:* An *int* variable that specifies the value of the *TID* bus.
    - *tdest:* An *int* variable that specifies the value of the *TDEST* bus.
  - **Description**
    - Sends data buffer.
- **genSingleTransfer()**
  - **Syntax**
    - *genSingleTransfer(tdata,tkeep,tstrb,tlast,tuser,tid,tdest)*
  - **Arguments**
    - *tdata:* An *256 bit* variable that specifies the value of the data bus.
    - *tkeep:* An *32 bit* variable that specifies the value of the *tkeep* bus.
    - *tstrb:* An *32 bit* variable that specifies the value of the *tstrb* bus.
    - *tlast:* An *int* variable that specifies the value of the *tlast* signal.
    - *tuser:* An *256 bit* variable that specifies the value of the *tuser* bus.
    - *tid:* An *8 bit* variable that specifies the value of the *tid* bus.
    - *tdest:* An *8 bit* variable that specifies the value of the *tdest* bus.
  - **Description**
    - Generate single transfer on the AXI4-Stream bus.
- **createUserBuf**()
  - **Syntax**
    - createUserBuf*(inBuf)*

    - **Arguments**
        - inBuff: *8 bit vector dynamic array* that contains user data buffer which should be transferred.
    - **Description**
        - Create the user defined buffer which will be transmitted alongside the data stream. The information of this buffer will be transmitted via *TUSER* bus when *sendData()* command is used. If the size of this buffer is less than transmitted data buffer the 0s will be transmitted.
- **busIdle()**
    - **Syntax**
        - *busIdle(idleCycles)*
    - **Arguments**
        - *idleCycles:* A *int* variable which specifies wait clock cycles
    - **Description**
        - Holds the bus in the idle state for the specified clock cycles
- **waitCommandDone()**
    - **Syntax**
        - waitCommandDone*()*
    - **Description**
        - Wait until all transactions in the buffer are finished
- **printStatus()**
    - **Syntax**
        - *printStatus()*
    - **Description**
        - Prints all errors occurred during simulation time.
- **startEnv()**
    - **Syntax**
        - *startEnv()*
    - **Description**
        - Start the AXI4-Stream master environment. Don't use data transfer commands before the environment start.

## Integration and Usage

The AXI4-Stream Master Verification IP integration into your environment is very easy. Instantiate the *axi4_str_m_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *axi4_str_m.sv* and *axi4_str_m_if.sv* files located inside the *axi4_str_vip/verification_ip* folder.

For usage the following steps should be done:

1. Import *AXI4STR_M* package into your test.
   - **Syntax**: *import AXI4STR_M::*;*
2. Create *AXI4STR_m_env* class object
   - **Syntax**: *AXI4STR_m_env axi4str=new(id_name,axi4_str_ifc_m,blockSize, userBitPerByte);*
   - **Description:** id_name is the name of the master vip(string variable), axi4_str_ifc_m is the reference to the AXI4-Stream Master Interface instance name. *blockSize* is the data bus size in bytes. Use only 1, 2, 4, 8, 16 and 32.
3. Start AXI4-Stream Master Environment.
   - **Syntax:** *axi4str.startEnv();*

This is all you need for AXI4-Stream master verification IP integration.

# 3. AXI4-Stream Slave

The AXI4-Stream Slave Verification IP models AXI4-Stream slave device.

## Commands

Commands marked as a *queued* will be put in the one queue and executed sequentially.

Commands marked as a *blocking* will block other commands execution until finishing its own process.

- **Configuration Commands**
  - **setRandDelay(): -** *non queued, non blocking*
  - **setTimeOut:-** *non queued, non blocking*
- **Data Processing Commands**
  - **getSingleTransfer(): -** *non queued, blocking*
  - **readData(): -** *non queued, blocking*
  - **readUserBuf():** - *non queued, non blocking*
  - **getTID_TDEST**(): - *non queued, non blocking*
- **Other Commands**
  - **startEnv(): -** *non blocking, should be called only once for current object*
  - **printStatus():** - *non queued, non blocking*

## Commands Description

All commands are *AXI4STR_s_env* class methods.

- **setRandDelay()**
    - **Syntax**
        - *setRandDelay(minBurst, maxBurst, minWait, maxWait)*
    - **Arguments**
        - *minBurst:* An *int* variable which specifies minimum value for burst length
        - *maxBurst:* An *int* variable which specifies maximum value for burst length
        - *minWait:* An *int* variable which specifies the minimum value for wait cycles
        - *maxWait:* An *int* variable which specifies the maximum value for wait cycles
    - **Description**
        - Enables/Disables bus random timings. To disable random timing all arguments should be set to zero.
- **setTimeOut()**
    - **Syntax**
        - *setTimeOut(readyTimeOut)*
    - **Arguments**
        - *readyTimeOut:* An *int* variable which specifies the maximum wait clock cycles for master response.
    - **Description**
        - Sets the maximum clock cycles for master response. If master delays response the error message will be generated.
- **getSingleTransfer()**
    - **Syntax**
        - *getSingleTransfer(tdata,tkeep,tstrb,tlast,tuser,tid,tdest)*
    - **Arguments**
        - *tdata:* An *256 bit* variable that specifies the value of the data bus.
        - *tkeep:* An *32 bit* variable that specifies the value of the *tkeep* bus.
        - *tstrb:* An *32 bit* variable that specifies the value of the *tstrb* bus.
        - *tlast:* An *int* variable that specifies the value of the *tlast* signal.
        - *tuser:* An *256 bit* variable that specifies the value of the *tuser* bus.
        - *tid:* An *8 bit* variable that specifies the value of the *tid* bus.
        - *tdest:* An *8 bit* variable that specifies the value of the *tdest* bus.
- **Description**

  - Get single transfer on the AXI4-Stream bus.
- **readData()**
  - **Syntax**
    - *readData(outBuff)*
  - **Arguments**
    - *outBuff: 8 bit vector array* that contains the read data packet.
  - **Description**
    - Reads the complete data packet from the bus.
- **readUserBuf()**
  - **Syntax**
    - *readUserBuf(outBuff)*
  - **Arguments**
    - *outBuff: 8 bit vector array* that contains the information from the user bus
  - **Description**
    - Read the user data.
- **getTID_TDEST()**
  - **Syntax**
    - *getTID_TDEST(tid, tdest)*
  - **Arguments**
    - *tid:* An *int* variable that contains the value from the *tid* bus.
    - *tdest:* An *int* variable that contains the value from the *tdest* bus.
  - **Description**
    - Returns the values from the *tdest* and *tid* buses. Use this command next to *readData()* to have the correct values.
- **startEnv()**
  - **Syntax**
    - *startEnv()*
  - **Description**
    - Starts AXI4-Stream slave environment.
- **printStatus()**
  - **Syntax**
    - *printStatus()*
  - **Description**

- Prints all errors occurred during simulation time.

## Integration and Usage

The AXI4-Stream Slave Verification IP integration into your environment is very easy. Instantiate the *axi4_str_s_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *axi4_str_s.sv* and *axi4_str_s_if.sv* files located inside the *axi4_stream_vip/verification_ip* folder.

For usage the following steps should be done:

1. Import *AXI4STR_S* package into your test.
   - **Syntax***: import AXI4STR_S::*;*
2. Create *AXI4STR_s_env* class object
   - **Syntax:** *AXI4STR_s_env axi4str = (id_name,axi4_str_ifc_s,blockSize, userBitPerByte)*
   - **Description:** id_name is the name of the slave vip(string variable), axi4_str_ifc_s is the reference to the AXI4-Stream Slave Interface instance name. *blockSize* is the data bus size in bytes. Use only 1, 2, 4, 8, 16 and 32.
- Start AXI4-Stream Slave Environment.
  - **Syntax:** *axi4str.startEnv();*

Now AXI4-Stream slave verification IP is ready to respond transactions initiated by master device.

# 4. Important Tips

In this section some important tips will be described to help you to avoid VIP wrong behavior.

## Master Tips

1. Call *startEnv()* task as soon as you create *AXI4STR_m_env* object before any other commands. You should call it not more then once for current object.
2. Before using Data Transfer Commands be sure that external hardware reset is done. As current release does not support external reset detection feature, the best way is to wait before DUT reset is done.

## Slave Tips

1. Call *startEnv()* task as soon as you create *AXI4STR_s_env* object before any other commands. It should be called before the first valid transaction initiated by AXI4-Stream master.