SysWip

Alternative Verification

AXI4-Lite Bus Verification IP User Manual

Release 1.0

Table of Contents

1. Introduction

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

Package Hierarchy

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

- axi4lite_vip
  - docs
  - examples
    - sim
    - testbench
  - verification_ip
    - master
    - slave

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-Lite Protocol
- Operates as a Master or Slave
- Supports 1, 2, 4, 8 and 16 bytes data block size
- Supports multiple outstanding transactions
- Programmable response type
- Read/Write response check
- Supports wait states injection
- Supports full random timings
- Supports misaligned transfers

Limitations

- Doesn't support awprot and arprot signals

2. AXI4-Lite Master

The AXI4-Lite Master Verification IP (VIP) initiates transfers on the AXI4-Lite 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**
  - **setRndDelay(): -** *non queued, non blocking*
  - **setTimeOut(): -** *non queued, non blocking*
  - **respReportMode**(): - *non queued, non blocking*
- **Data Transfer Commands**
  - **writeData(): -** *queued, non blocking*
  - **readData(): -** *queued, non blocking*
  - **getData**(): - *queued, blocking*
  - **pollData**(): - *queued, blocking*
  - **getWrResp(): -** *queued, blocking*
  - **wrAddrChannelIdle(): -** *queued, non blocking*
  - **wrDataChannelIdle**(): - *queued, non blocking*
  - **rdAddrChannelIdle**(): - *queued, non blocking*
- **Other Commands**
  - **startEnv(): -** *non queued, non blocking*
  - **wrAddrChannelDone():-** *queued, blocking*
  - **wrDataChannelDone():-** *queued, blocking*
  - **rdAddrChannelDone():-** *queued, blocking*
  - **rdDataChannelDone():-** *queued, blocking*
  - **wrRespChannelDone():-** *queued, blocking*
  - **printStatus():-** *non queued, non blocking*

Commands Description

All commands are *AXI4Lite_m_env* class methods.

- **setRndDelay()**
  - **Syntax**
    - *setRndDelay(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, pollTimeOut)*
  - **Arguments**
    - *readyTimeOut:* An *int* variable which specifies the maximum wait clock cycles for slave response.
    - *pollTimeOut:* An *int* variable which specifies maximum wait clock cycles for polling.
  - **Description**
    - Sets the maximum clock cycles for slave slave response and polling. If slave delays response the error message will be generated.
- **respReportMode()**
  - **Syntax**
    - *respReportMode(respReportEn)*
  - **Arguments**
    - *respReportEn:* An *int* variable which specifies the read/write response report mode.
  - **Description**
    - Set Read/Write response report mode. If *respReportEn* is 1 all not OK responses will be detected on the fly and reported by "printStatus" function.

- **writeData()**
  - **Syntax**
    - *writeData(wrRespPtr, addr, inBuff)*
  - **Arguments**
    - *addr:* An *int* variable that specifies the start byte address
    - *inBuff:* 8 bit vector queue that contains data buffer which should be transferred
    - *wrRespPtr:* An *int* variable that specifies the write response buffer pointer
  - **Description**
    - Writes data buffer. Returns pointer where will be located write response buffer.
- **readData()**
  - **Syntax**
    - *readData(addr, dataLength, dataOutPtr)*
  - **Arguments**
    - *addr:* An *int* variable that specifies the start byte address
    - *dataLength:* An *int* variable that specifies the amount of bytes which should be read.
    - *dataOutPtr*: An *int* variable that specifies the read data buffer pointer
  - **Description**
    - Generate read transactions. Returns pointer where will be located read data buffer.
- **getData()**
  - **Syntax**
    - *getData(rdDataPtr, outBuff, outRespBuff)*
  - **Arguments**
    - *rdDataPtr:* An *int* variable that specifies the data buffer read pointer
    - *outBuff:* 8 bit vector queue that contains read data buffer
    - *outRespBuff*: 8 bit vector queue that contains read response buffer
  - **Description**
    - Returns read data and response buffers from specified location.

- **getWrResp()**
  - **Syntax**
    - *getWrResp(wrRespPtr, outRespBuff)*
  - **Arguments**
    - *wrRespPtr:* An *int* variable that specifies the write response buffer pointer
    - *outRespBuff:* 8 bit vector queue that contains write response buffer
  - **Description**
    - Returns write response buffer from specified location.
- **pollData()**
  - **Syntax**
    - *pollData(addr, pollData)*
  - **Arguments**
    - *addr:* An *int* variable that specifies the start address
    - *pollData:* 8 bit vector queue that contains data buffer which should be compared with read data buffer
  - **Description**
    - Read data buffer starting from *addr* and compare it with *pollData* buffer. Repeat until buffers are equal or until time out occurred.
- **wrAddrChannelIdle()**
  - **Syntax**
    - *wrAddrChannelIdle(idleCycles)*
  - **Arguments**
    - *idleCycles:* A *int* variable which specifies wait clock cycles
  - **Description**
    - Holds the write address channel in the idle state for the specified clock cycles
- **wrDataChannelIdle()**
  - **Syntax**
    - *wrDataChannelIdle(idleCycles)*
  - **Arguments**
    - *idleCycles:* A *int* variable which specifies wait clock cycles
  - **Description**

    Holds the write data channel in the idle state for the specified clock cycles

- **rdAddrChannelIdle()**
  - **Syntax**
    - *rdAddrChannelIdle(idleCycles)*
  - **Arguments**
    - *idleCycles:* A *int* variable which specifies wait clock cycles
  - **Description**
    - Holds the read address channel in the idle state for the specified clock cycles
- **wrAddrChannelDone()**
  - **Syntax**
    - *wrAddrChannelDone()*
  - **Description**
    - Wait until all transactions in the write address channel buffer are finished
- **wrDataChannelDone()**
  - **Syntax**
    - *wrDataChannelDone()*
  - **Description**

    Wait until all transactions in the write data channel buffer are finished
- **rdAddrChannelDone()**
  - **Syntax**
    - *rdAddrChannelDone()*
  - **Description**

    Wait until all transactions in the read address channel buffer are finished
- **rdDataChannelDone()**
  - **Syntax**
    - *rdDataChannelDone()*
  - **Description**

    Wait until all transactions in the read data channel buffer are finished
- **wrRespChannelDone()**
  - **Syntax**
    - *wrRespChannelDone()*
  - **Description**

    Wait until all transactions in the write response channel buffer are finished

- **printStatus()**
  - **Syntax**
    - *printStatus()*
  - **Description**
    - Prints all errors occurred during simulation time and returns error count.
- **startEnv()**
  - **Syntax**
    - *startEnv()*
  - **Description**
    - Starts AXI4-Lite master environment. Don't use data transfer commands before the environment start.

Integration and Usage

The AXI4-Lite Master Verification IP integration into your environment is very easy. Instantiate the *axi4lite_m_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *axi4lite_m.sv* and *axi4lite_m_if.sv* files located inside the *axi4lite_vip/verification_ip/master* folder.

For usage the following steps should be done:

1. Import *AXI4LITE_M* package into your test.
   - **Syntax***: import AXI4LITE_M::*;*
2. Create *AXI4Lite_m_env* class object
   - **Syntax***: AXI4Lite_m_env axi4lite= new(id_name, axi4lite_ifc_m, dataSize);*
   - **Description:** *id_name* is the name of the master vip(*string* variable), *axi4lite_ifc_m* is the reference to the AXI4-Lite Master Interface instance name. *dataSize* is data block size in bytes. Use only 1, 2, 4, 8 and 16.
3. Start AXI4-Lite Master Environment.
   - **Syntax:** *axi4lite.startEnv();*

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

3. AXI4-Lite Slave

The AXI4-Lite Slave Verification IP models AXI4-Lite slave device. It has an internal memory which is accessible by master device as well as by corresponding commands.

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**
  - **setRndDelay(): -** *non queued, non blocking*
  - **setWrResp():-** *non queued, non blocking*
  - **setRdResp():**- *non queued, non blocking*
  - **setMemCleanMode():** - *non queued, non blocking*
- **Data Processing Commands**
  - **putData(): -** *non queued, non blocking*
  - **getData(): -** *non queued, non blocking*
  - **pollData():** - *non queued, blocking*
- **Other Commands**
  - **startEnv(): -** *non blocking, should be called only once for current object*
  - **printStatus():** - *non queued, non blocking*

Commands Description

All commands are *AXI4Lite_s_env* class methods.

- **setRndDelay()**
  - **Syntax**
    - *setRndDelay(minAckDelay, maxAckDelay)*
  - **Arguments**
    - *minAckDelay:* A *int* variable that specifies minimum value for response delay
    - *maxAckDelay:* A *int* variable that specifies maximum value for response delay
  - **Description**
    - Enables/Disables slave response random delays. To disable set all arguments to zero.
- **setWrResp()**
  - **Syntax**
    - *setWrResp(address, wrResp)*
  - **Arguments**
    - *address:* A 32 *bit vector* which specifies response address
    - *wrResp:* A *int* which specifies the slave write response type.
  - **Description**

        - Sets slave write response type for the specified address. Specified address will be block aligned internally.

- **setRdResp()**
    - **Syntax**
        - *setRdResp(address, rdResp)*
    - **Arguments**
        - *address:* A 32 *bit vector* which specifies response address
        - *rdResp:* A *int* which specifies the slave read response type.
    - **Description**
        - Sets slave read response type for the specified address. Specified address will be block aligned internally.
- **setMemCleanMode()**
    - **Syntax**
        - *setMemCleanMode(memClean)*
    - **Arguments**
        - *memClean:* An *int* variable which specifies internal memory clean mode
    - **Description**
        - Sets internal memory clean mode. 0 – no clean. 1- Only master read transactions will clean the memory cell. 2 – Only *getData* command will clean the memory cell. 3 – Both master read transactions and *getData* function will clean the memory cell.
- **putData()**
    - **Syntax**
        - *putData(startAddr, dataInBuff*)
    - **Arguments**
        - *startAddr:* An *int* variable that specifies the start byte address
        - *dataInBuff:* 8 bit vector queue that contains data buffer which will be written to the memory.
    - **Description**
        - Puts data buffer to the to the internal memory
- **getData()**
    - **Syntax**
        - *getData(startAddr, dataOutBuff, lenght)*
    - **Arguments**
        - *startAddr:* An *int* variable that specifies the start byte address

        - *dataOutBuff:* 8 bit vector queue that contains read data from memory.
        - *length:* An *int* variable that specifies the amount of bytes which will be read from the internal memory.
    - **Description**
        - Reads data from the internal memory.
- **pollData()**
    - **Syntax**
        - *pollData(address, pollData, pollTimeOut)*
    - **Arguments**
        - *address:* An *int* variable that specifies the start address
        - *pollData:* 8 bit vector queue that contains data buffer which should be compared with read data buffer
        - *pollTimeOut:* An *int* variable that specifies the poll time out clock cycles.
    - **Description**
        - Read data buffer from internal memory and compare it with *pollData* buffer. Repeat until buffers are equal or until time out occurred.
- **startEnv()**
    - **Syntax**
        - *startEnv()*
    - **Description**
        - Starts AXI4lite slave environment.
- **printStatus()**
    - **Syntax**
        - *printStatus()*
    - **Description**
        - Prints all errors occurred during simulation time and returns error count.

Integration and Usage

The AXI4-Lite Slave Verification IP integration into your environment is very easy. Instantiate the *axi4lite_s_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *axi4lite_s.sv* and *axi4lite_s_if.sv* files located inside the *axi4lite_vip/verification_ip/slave* folder.

For usage the following steps should be done:

1. Import *AXI4LITE_S* package into your test.
    - **Syntax***: import AXI4LITE_S::*;*
2. Create *AXI4Lite_s_env* class object

- **Syntax:** *AXI4Lite_s_env_s_env axi4lite = new(id_name , axi4lite_ifc_s, dataSize);*
- **Description:** *id_name* is the name of the slave vip(*string* variable), *axi4lite_ifc_s* is the reference to the AXI4-Lite Slave interface instance name. *dataSize* is data block size in bytes. Use only 1, 2, 4, 8 and 16.

3. Start AXI4-Lite Slave Environment
    - **Syntax:** *axi4lite.startEnv();*

Now AXI4-Lite slave verification IP is ready to respond transactions initiated by master device. Use data processing commands to put or get data from the internal memory.

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 *AXI4Lite_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 *AXI4Lite_s_env* object before any other commands. It should be called before the first valid transaction initiated by AXI4-Lite master.