SysWip

Alternative Verification

MDIO Serial Bus Verification IP User Manual

Release 1.0

Table of Contents

1. Introduction

The MDIO Verification IP described in this document is a solution for verification of MDIO serial bus master(STA) and slave(MMD) devices. The provided MDIO verification package includes master and slave verification IPs and examples. It will help engineers to quickly create verification environment and test their MDIO master and slave devices.

Package Hierarchy

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

- mdio_vip
    - docs
    - examples
        - sim
        - testbench
    - verification_ip

The Verification IP is located in the *verification_ip* folder.

Features

- Easy integration and usage
- Free SystemVerilog source code
- Compliant to the MDIO protocol specified by the IEEE 802.3 standard "Clause 22"
- Supports extended operation mode defined in the 802.3ae standard "Clause 45"
- Operates as a Master or Slave
- Supports multiple slaves
- Supports wait states injection
- Supports full random timings

2. MDIO Master

The MDIO Master Verification IP (VIP) initiates transfers on the MDIO serial 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*

- **setST(): -** *non queued, non blocking*

- **Data Transfer Commands**
  - **writeAddr(): -** *queued, non blocking*
  - **writeWord(): -** *queued, non blocking*
  - **readWord(): -** *queued, non blocking*
  - **getWord**(): - *queued, blocking*
  - **pollWord**(): - *queued, blocking*
  - **setIdle():** - *queued, non blocking*
- **Other Commands**
  - **startEnv(): -** *non queued, non blocking*
  - **mdioDone()*: -*** *non queued, non blocking*
  - **printStatus():-** *non queued, non blocking*

Commands Description

All commands are *MDIO_m_env* class methods.

- **setRndDelay()**
  - **Syntax**
    - *setRndDelay(minDelay, maxDelay)*
  - **Arguments**
    - *minDelay:* An *int* variable which specifies minimum clock cycles between frames.
    - *maxDelay:* An *int* variable which specifies maximum clock cycles between frames.
  - **Description**
    - Enables/Disables bus random timings delays between frames. To disable random timing all arguments should be set to zero.
- **setST()**
  - **Syntax**
    - *setST(st)*
  - **Arguments**
    - *st:* An *int* variable which specifies the ST mode.
  - **Description**
    - Sets the mode. ST == 0 for Clause 45 or ST == 1 for Clause 22

- **writeAddr()**
  - **Syntax**
    - *writeAddr(phyAddr, devType, addrData)*
  - **Arguments**
    - *phyAddr:* An *int* variable that specifies the PHY address
    - *devType:* An *int* variable that specifies the device type
    - *addrData:* An *int* variable that specifies the write address word
  - **Description**
    - Writes address word. Should only be used in Clause 45 mode
- **writeWord()**
  - **Syntax**
    - *writeWord(phyAddr, devType, data16)*
  - **Arguments**
    - *phyAddr:* An *int* variable that specifies the PHY address
    - *devType:* An *int* variable that specifies the device type
    - *data16:* An *int* variable that specifies the write data word
  - **Description**
    - Writes data word
- **readWord()**
  - **Syntax**
    - *readWord(dataOutPtr, phyAddr, devType, incr)*
  - **Arguments**
    - *phyAddr:* An *int* variable that specifies the PHY address
    - *devType*: An int variable that specifies the device type
    - *incr:* An *int* variable that specifies the read increment mode
    - *dataOutPtr*: An *int* variable that specifies the buffer pointer where should be stored the read data word
  - **Description**
    - Generate read transactions. Returns pointer where will be located the read data word.
- **getWord()**
  - **Syntax**
    - *getWord(rdDataPtr, data16)*

- **Arguments**
  - *rdDataPtr:* An *int* variable that specifies the data buffer read pointer
  - *data16:* An *int* variable that specifies the read data word
- **Description**
  - Returns read data word from specified location.

- **pollWord()**
  - **Syntax**
    - *pollWord(phyAddr, devType, addr, pollDataWord, pollTimeOut)*
  - **Arguments**
    - *phyAddr:* An *int* variable that specifies the PHY address
    - *devType*: An int variable that specifies the device type
    - *addr:* An int variable that specifies the poll address
    - *pollDataWord :* An int variable that specifies the poll data
    - *pollTimeOut:* An int variable that specifies the poll time out cycles
  - **Description**
    - Poll specified addresses until read data word is equal to *pollDataWord* word. If poll counter is reached to "pollTimeOut" value stop polling and generate error message. In Clause 22 mode the *addr* is ignored and the *devType* is used as a polling address.
- **setIdle()**
  - **Syntax**
    - *setIdle(idleCycles)*
  - **Arguments**
    - *idleCycles:* A *int* variable which specifies wait clock cycles
  - **Description**
    - Holds the MDIO serial bus in the idle state for the specified clock cycles
- **mdioDone()**
  - **Syntax**
    - *mdioDone()*
  - **Description**

    Wait until all transactions in the input command buffer are finished
- **printStatus()**
  - **Syntax**

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

Integration and Usage

The MDIO master verification IP integration into your environment is very easy. Instantiate the *mdio_m_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *mdio_m.sv* and *mdio_m_if.sv* files located inside the *mdio_vip/verification_ip/* folder.

For usage the following steps should be done:

1. Import *MDIO_M* package into your test.
    - **Syntax**: *import MDIO_M::*;*
2. Create *MDIO_m_env* class object
    - **Syntax**: *MDIO_m_env mdio_m= new(mdio_ifc_m);*
    - **Description:** *mdio_ifc_m* is the reference to the MDIO master interface instance name.
3. Start MDIO Master Environment.
    - **Syntax:** *mdio.startEnv();*

This is all you need for MDIO master verification IP integration.

3. MDIO Slave

The MDIO Slave Verification IP models MDIO slave(MMD) 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**
    - **setAddr(): -** *non queued, non blocking*

  - **setMemCleanMode():** *- non queued, non blocking*
- **Data Processing Commands**
  - **putWord(): -** *non queued, non blocking*
  - **getWord(): -** *non queued, non blocking*
  - **pollWord():** *- 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 *MDIO_s_env* class methods.

- **setAddr()**
  - **Syntax**
    - *setAddr(phyAddr, devType)*
  - **Arguments**
    - *phyAddr:* An *int* variable which specifies PHY address
    - *devType:* An *int* variable which specifies device type
  - **Description**
    - Sets slave device PHY address and device type. Slave device will respond master's transaction only when PHY address and device type transmitted by master are equal to the slave PHY address and device type.
- **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 *getWord* command will clean the memory cell. 3 – Both master read transactions and *getWord* function will clean the memory cell.
- **putWord()**
  - **Syntax**
    - *putWord(addr, dataWord*)

- **Arguments**
  - *addr:* An *int* variable that specifies the data word address
  - *dataWord:* An *int* variable that specifies the data word
- **Description**
  - Puts data word to the to the internal memory

- **getWord()**
  - **Syntax**
    - *getWord(addr, dataWord)*
  - **Arguments**
    - *addr:* An *int* variable that specifies the data word address
    - *dataWord:* An *int* variable that specifies the read data word
  - **Description**
    - Reads data from the internal memory.
- **pollWord()**
  - **Syntax**
    - *pollData(addr, dataWord)*
  - **Arguments**
    - *addr:* An *int* variable that specifies the poll address
    - *dataWord:* An *int* variable that specifies the poll data word
  - **Description**
    - Polls the internal memory address until the read data is equal to the *dataWord*.
- **startEnv()**
  - **Syntax**
    - *startEnv()*
  - **Description**
    - Starts MDIO slave environment.
- **printStatus()**
  - **Syntax**
    - *printStatus()*
  - **Description**
    - Prints all errors occurred during simulation time and returns error count.

Integration and Usage

The MDIO Slave Verification IP integration into your environment is very easy. Instantiate the *mdio_s_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *mdio_s.sv* and *mdio_s_if.sv* files located inside the *mdio_vip/verification_ip/* folder.

For usage the following steps should be done:

1. Import *MDIO_S* package into your test.
   - **Syntax**: *import MDIO_S::*;*
2. Create *MDIO_s_env* class object
   - **Syntax:** *MDIO_s_env mdio_s = new(mdio_ifc_s);*
   - **Description:** *mdio_ifc_s* is the reference to the MDIO Slave interface instance name.
3. Start MDIO Slave Environment
   - **Syntax:** *mdio_s.startEnv();*

Now MDIO 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 *MDIO_m_env* object before any other commands.
2. The MDIO master generates MDIO output clock (MDC). The frequency of that clock is equal to the system clock(*clk* signal in the *mdio_ifc_m* module) frequency. To avoid any race conditions always connect MDC clock generated by master VIP to the slave devices.
3. 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 *MDIO_s_env* object before any other commands. It should be called before the first valid transaction initiated by MDIO master.
2. Set PHY address and device type before any transactions. If MDIO bus has multiple slaves use different PHY address and device type for each slave.