BASIC SYSTEM MMU

TestBench Description

Document Version	1.0
Release Date	May 17th, 2016

Version: 1.0

Table of Contents

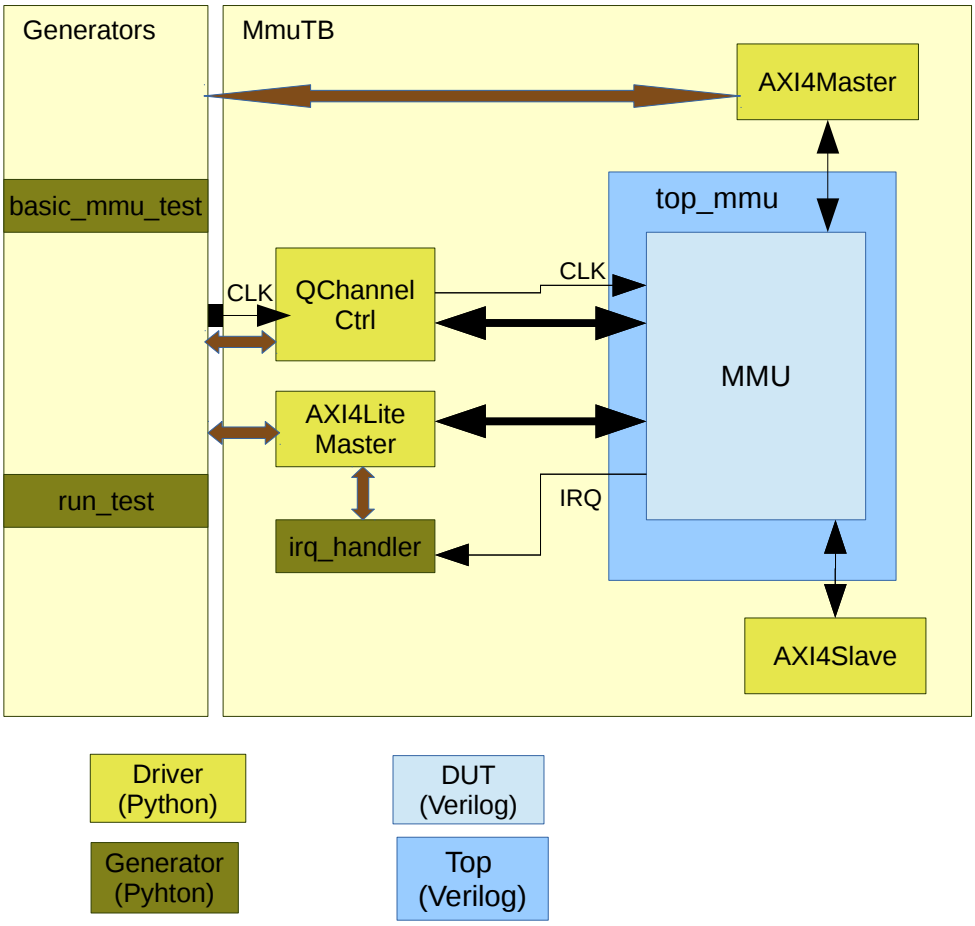
1 Overview

1.1 Introduction

MMU verification environment is based on cocotb environment, and so developed in Python.

For further reading on cocotb, please refer cocotb documentation (http://cocotb.readthedocs.io/en/latest/index.html)

Below figure shows testbench hierarchy:



2 Components description

2.1 Top verilog

Language : Verilog

File location : test/top_mmu.v

All inputs vectors from MmuTB are delayed before being provided to DUT.

All outputs vectors form DUT are delayed before being provided to MMUTB

Objective is to avoid any signals race condition.

2.2 Drivers / BFM

Language : Python

File location: cocotb/driver/amba.py

That file has been updated vs standard cocotb delivery, so the original amba.py has to be replaced by the one provided.

- AXI4LiteMaster: Control MMU programming interface and verify protocol.
- AXI4Slave: Interface with MMU downstream port and verify protocol. Embeds a memory whom size is configurable
- AXI4Master: Control MMU upstream port and verify protocol.
- QchannelCtrl: Interface with MMU low power interface.

Refer to source file to understand how to fine tune the various parameters.

2.3 Generators

Language : Python

File location: test/test_mmu.v

Generators control and respond to DUT via drivers.

2.3.1 irq_handler

- In case of IRQ active (high level), reads and prints FAR and FAR2 registers.
- It check has well that IRQ is reset after reading FAR2 register.

2.3.2 basic_mmu_test

- Check default value of control registers

- MMU Bypassed: Write / Read access to slave memory
- write MMU translation table (user space)
- write MMU translation table (kernel space)
- program MMU translation table
- MMU enabled: Unsecure, user space Write / Read access
- Bypass MMU, and check that data have been written to the targeted PA address
- MMU enabled: Unsecure, user space Write / Read access with bad stream ID. Access is expected to fail and test is aborted in case not.
- Check that control registers can not be written in user mode.
- Check that secure control registers can not be written in non-secure mode

2.3.3 run_test

Random MMU test having an number of constraint random parameters that are controlled by the cocotb TestFactory feature.

Cocotb TestFactory allows to generate a bench of run_test according to those parameters making sure that all combinations are simulated.

Constraint Random Parameters : Controlled by TestFactory

==========

Translation Table parameters:

---

table_count: Nbre of level page tables (1 => 1GB, 2 => 2MB, 3 => 4KB)

sec: Secure Not Secure

ap: Access Permission

control register parameters:

---

a1: For Kernel space, select ASID1 when equal to 1 and ASID0 when equal to 0

Test parameters:

---

access_number: Number of repeated sequences

Timing parameters:

---

address_data_latency: Controls relative timing of Address channel Vs Data channel of AXI4LiteMaster and AXI4Master

0 => address and data channel start

<0 => address channel start after data channel. Exact delay is random

>0 => address channel start before data channel. Exact delay is random

slave_awrready_latency: Controls latency of AXI4Slave awready signal

0 => No Latency

1 => Random latency

slave_rvalid_latency: Controls latency of AXI4Slave rvalid signal

0 => No Latency

1 => Random latency

master_rready_latency: Controls latency of AXI4Master rready signal

0 => No Latency

1 => Random latency

master_wvalid_latency: Controls latency of AXI4Master wvalid signal

0 => No Latency

1 => Random latency

```
    Random Parameters: (not controlled by Test Factory)
    ===================================
    AXI Master Burst length
    user and kernel L1 table Physical address
    user and kernel block Physical address
    Stream Matching Register and AxID attribute
    Virtual address (from AXI4Master address channel)
    Page index (Only if table_count = 3)
    """
```