GPIO

Verification Environment and Test Plan

Table of Contents

List of Figures

List of Tables

1.INTRODUCTION

1.1 Document Overview

This document describes the Verification environment and Test plan for the GPIO. It identifies the various components that form the GPIO Verification Environment and defines the test bench features and test scenarios. GPIO is a block in the PULPino project which is used to connect the particular peripheral device with the core

1.2 Terms and Definitions

Table 1.2.1: Terms and Definitions

Abbreviation / Term	Expansion / Definition
IP	Intellectual property
ASIC	Application Specific Integrated Circuit
GPIO	General Purpose Input Output
UVM	Universal Verification Methodology
DUT	Design Under Test
MON	Monitor
TB	Test Bench
TC	Test Case

2. TEST BENCH OVERVIEW

2.1 Verification Objective

The primary objective is to develop a verification environment for the GPIO, to perform a block level verification of each component with possible test cases using UVM Methodology. The idea of using the Universal verification methodology [UVM] is to preserve a single environment for block level verification.

2.2 Testbench Features

- UVM based Verification Environment
- Constrained random data generation
- Scoreboard to check data integrity

2.3 Verification Architecture

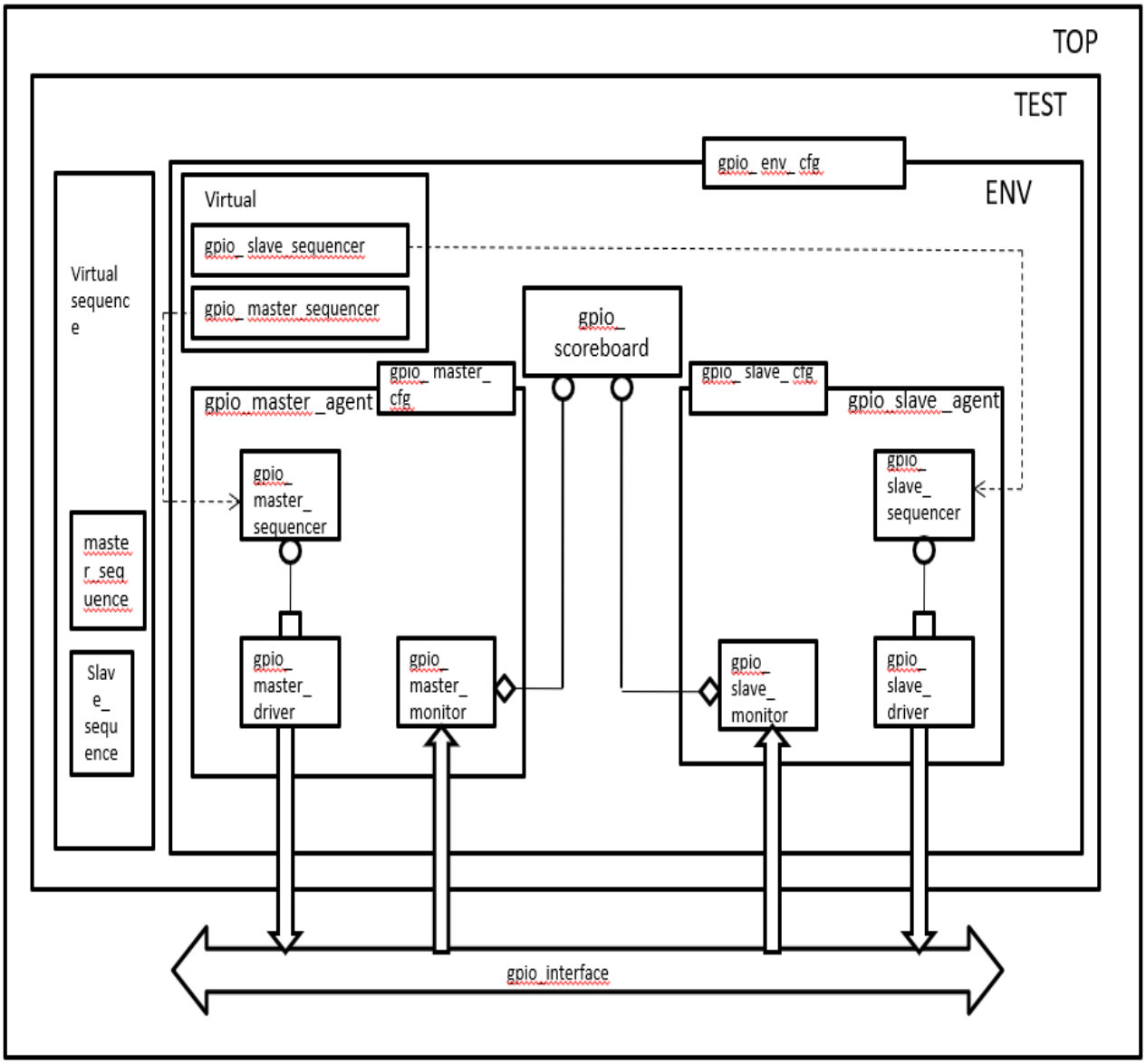

Figure 2.1: GPIO Verification Environment Architecture

The basic architecture for UVM verification is shown in figure 2.1. The most important and basic element in UVM architecture is agent.

2.3.1 TOP MODULE:

Set the configuration with virtual interface. Default method run test () will calls the all phases and runs the test cases. In the top module instantiate DUT signals with interface signals.

2.3.2 TEST:

Test is a place where we start the sequences on sequencer. As in sequences here also we have a base test that will be extended from unmutes. The further child tests will be made from this base test class. In base test we will set all the parameters of configuration database class according to our requirements. We will also get the interface set in top and again set it to the interface handles in local configuration database classes.

The base test also creates the environment. All these things happen in build phase of base test. In child test we just create the handle of sequence and start it on some sequencer, before starting the sequence an objection is raised and the same objection is dropped after starting sequence. This raise and drop of objection is done so that one who is simulating could know when the test bench comes out of the run phase.

2.3.3 ENV:

This class is created in test and is also a most important component of our test bench. Environment creates agent-tops, scoreboard, virtual sequencer (if required) etc. The environment makes connection between monitors and scoreboard. In case one has virtual sequencer, one has to connect physical sequencers with the handles of physical sequencers in virtual sequencer. Here we also set the local configuration database class parameters.

2.3.4 VIRTUAL SEQUENCE:

Virtual sequence is also extended from uvm_sequence as normal sequences but is parameterized by uvm_sequence_item. It has handles of both master and slave sequences written, handles of physical sequencers and virtual

sequencer. This class has a base sequence which will connect handles of physical sequencers to the handles of physical sequencers in virtual sequencer.

In child sequences one has to create handles of physical sequence which is to be started and after creating we start it. This virtual sequence handle is made in test and then it is created and then started. The start from test calls this start and these starts go to respective physical sequence and start that sequence.

3. Test Bench Components

3.1 Score board

Scoreboards are used for checking the data integrity. Expected data is collected from the respective monitor and compared with the actual data coming from peripheral monitor

3.2 Virtual Sequencer

Virtual sequencer is extended from uvm_sequencer and is parameterized by uvm_sequence_item. This is parameterized by uvm_sequence_item because this sequencer has to take both type of data i.e. master and slave.

This sequencer is required because we cannot run virtual sequences on physical sequencers so we make a virtual sequencer and run sequences on it. This sequencer is pointed to parent type handle of sequencer in virtual sequence using $cast. The virtual sequencer has the handles of physical sequencers which are pointed to physical sequencers in the environment.

3.3 Interface

The Interface construct in System Verilog is used specifically to encapsulate the communication between blocks. This is a System Verilog language construct used to connect the DUT to the Testbench.

3.4 Agents

Active agents were used here. It defines whether the sequencer, driver and monitor are to be created or not, using an enum which is defined in UVM Factory i.e. uvm_active_passive_enum. If the value of this enum is equal to is active all

their sub-blocks i.e. driver, sequencer and monitor are created but if value is equal to is passive only monitor is created. This enum can be set from the test case.

This agent also connects the sequencer and driver during connect phase. In this project there are two agents i.e. master agent and slave agent, which will create their own driver, sequencer and monitor. Both are active agents

3.4.1 Sequencer

The sequencer in UVM just acts as a bridge between sequence and driver. This is the only "non-virtual" class in our UVM factory. The sequencer is also parameterized by transaction class. The sequencer takes the randomized data from sequences and passes it to the driver to which it is connected, thus it connects several sequences to driver. The sequencers for both master and salve have same functionality.

3.4.1.1 Sequence (Transaction class)

Sequence will get the request from the driver through Sequencer and then will randomize the signals send the transaction to the driver through sequencer.

3.4.2 Driver

Driver sends request to the sequencer. Driver will get the transaction from the sequencer. It will convert the transaction level signals to pin level signals. It will drive input stimulus to the DUT based upon protocol through the virtual interface.

3.4.2.1 gpio_master_driver

Based on the register direction drive the data into slave (pin should act as input). based upon the test case need to follow the test procedure

3.4.2.2 gpio_slave_driver

Based on the register direction drive the data into master (same pin should act as output). based upon the test case need to follow the test procedure

3.4.3 Monitor

The Monitor as name specified monitors the data from the interface. The Monitor in UVM is extended from uvm_monitor. Monitor has analysis port declared and created during build phase. Monitor gets the interface using local configuration database class. In run phase we monitor the data according to protocol. In monitor has write method, through this method monitor will send the data to scoreboard.

3.4.3.1 gpio_master_monitor

Collected values from the master driver and connect to the scoreboard for the data comparison

3.4.3.2 gpio_slave_monitor

Collected values from the slave driver and connect to the scoreboard for the data comparison

4 GPIO Signal Description

Table 4.1: GPIO Signal Description

Signal	Direction	Description
gpio_in [31:0]	output	Transmit data
gpio_out [31:0]	input	Receive data
gpio_dir [31:0]	input	Request to send
pad_cfg [5:0] [31:0]	input	Pad configuration
interrupt	input	Interrupt

5 File Structure

Table 5.1 : File Structure

directory	/pdtools/workarea/frontend/deepika/new_gpio_vip			
	new_gpio_vip	gpio_top/gpio_test/gpio_env/sim/gpio_master_agent/gpio_slave_agent		
		sim	makefile	Containing the run scripts for running simulation and regression
		gpio_top	gpio_top.sv	Instantiation and set the Configuration
			gpio_if.sv	Interface signals
		gpio_test	gpio_test.sv	Starting the sequence
		gpio_env	gpio_scoreboard	Comparing the data
			gpio_virtual_sequnce	uvm codes of virtual sequence
			gpio_env.sv	Environment code
			gpio_package.sv	List of files
			gpio_env_config.sv	Config parameters for GPIO environment
			gpio_virtual_sequencer.sv	uvm codes of virtual sequencer
		gpio_master_agent	gpio_master_agent.sv	agent source code
			gpio_master_driver.sv	Driver source code
			gpio_master_monitor.sv	Monitor source code
			gpio_master_sequencer.sv	Sequencer source code
			gpio_master_sequence.sv	Sequence item
			gpio_master_config.sv	Config parameters

		gpio_slave_agent	gpio_slave_agent.sv	agent source code
			gpio_slave_driver.sv	Driver source code
			gpio_slave_monitor.sv	Monitor source code
			gpio_slave_sequencer.sv	Sequencer source code
			gpio_slave_sequence.sv	Sequence item
			gpio_slave_config.sv	Config parameters

S