

Design Verification Strategy

For APB Timer

**Rev. A**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **REV** | **DESCRIPTION** | **AUTHOR** |
| 19-03-2024 | A | First Draft | Pooja C |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

**Contents**

[Revision History 1](#_heading=h.gjdgxs)

[Table of Contents 2](#_heading=h.30j0zll)

[List of Figures and Tables 3](#_heading=h.1t3h5sf)

[0](#_heading=h.4d34og8) Scope 4

[1](#_heading=h.2s8eyo1) DUT Introduction 4

1.1 APB Timer description

[1.2](#_heading=h.17dp8vu)  I/O diagram

[1.3](#_heading=h.17dp8vu)  Registers

2 Features

3 Test Bench Description

3.1 Test Bench overview

3.2 APB Timer Test Bench architecture diagram

7 Functional coverage

**List of Figures**

**Figure No** **Figure Name**

1 APB Timer Module Block Diagram

2 APB Test bench architecture

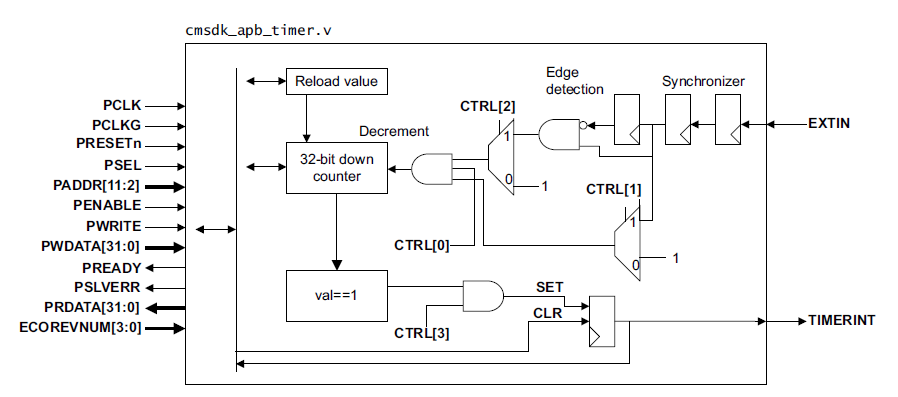
**List of Tables:**

**Table No Table Name**

1 Programmers model

**Introduction:**

In VLSI (Very Large Scale Integration) design, an APB (Advanced Peripheral Bus) timer typically refers to a timer module interfaced with the Advanced Peripheral Bus (APB) in an ARM-based microcontroller or System-on-Chip (SoC). The APB is a low-speed peripheral bus used to connect various peripheral devices to the microcontroller or SoC.

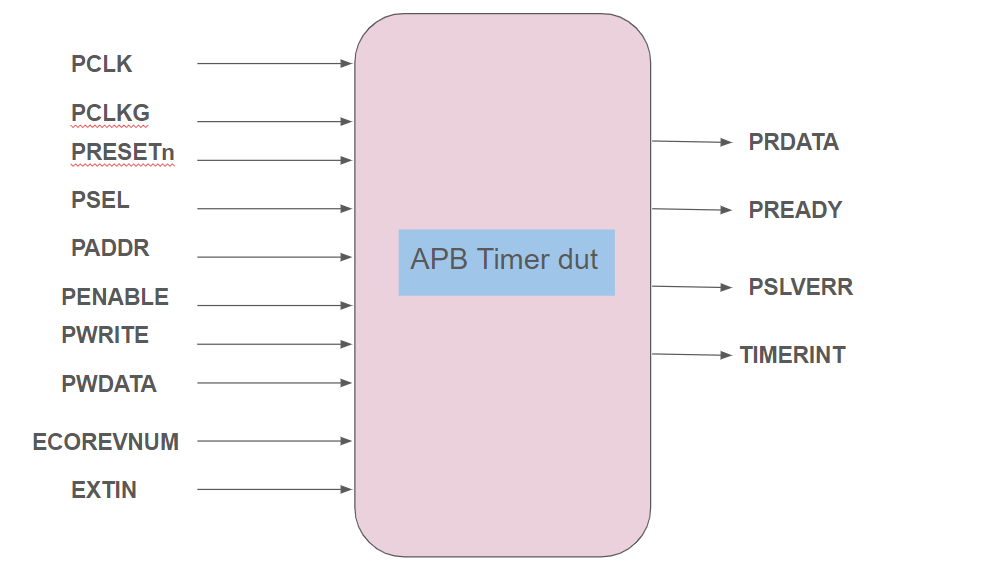
****

**Figure 1. APB Timer**

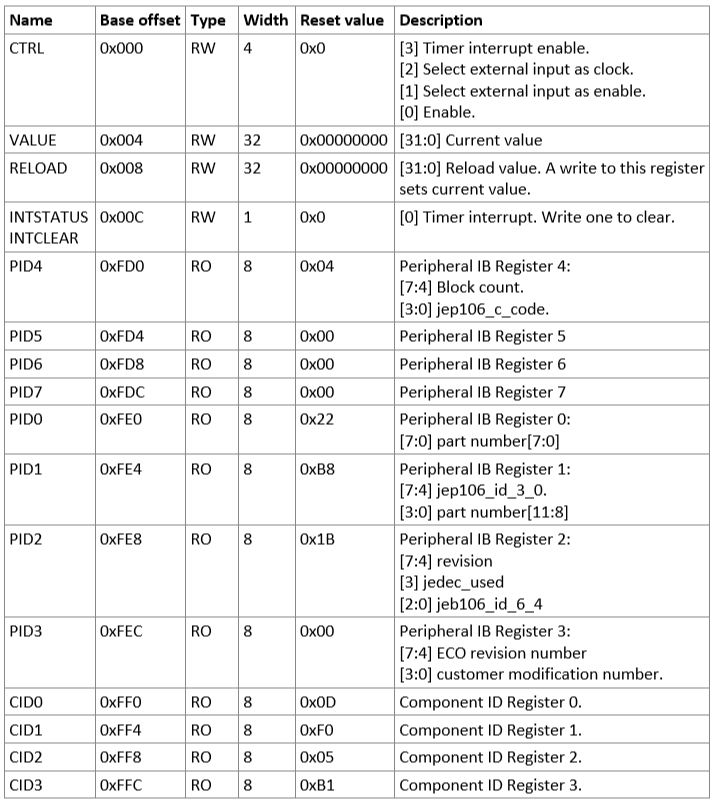
**Description:**

The APB timer, cmsdk\_apb\_timer.v, is a 32-bit down-counter with the following features:

* You can generate an interrupt request signal, **TIMERINT**, when the counter reaches 0.
* The interrupt request is held until it is cleared by writing to the INTCLEAR Register.
* You can use the zero to one transition of the external input signal, **EXTIN**, as a timer enable.
* If the APB timer count reaches 0 and, at the same time, the software clears a previous interrupt status, the interrupt status is set to 1.
* The external clock, **EXTIN**, must be slower than half of the peripheral clock because it is sampled by a double flip-flop and then goes through edge-detection logic when the external inputs act as a clock.
* A separate clock pin, **PCLKG**, for the APB register read or write logic that permits the clock to peripheral register logic to stop when there is no APB activity.
* Component ID and Peripheral ID Registers. These read-only ID registers are optional.
  + You must modify the following in these registers:
  + Part number, 12 bits.
  + JEDEC ID value, 7 bits.
* The **ECOREVNUM** input signal is connected to the ECO revision number in Peripheral ID Register 3.

****

**Figure 2. IO Diagram**

****

**Table 1: Programmers model**

**Test Bench Description.**

**APB Timer environment**

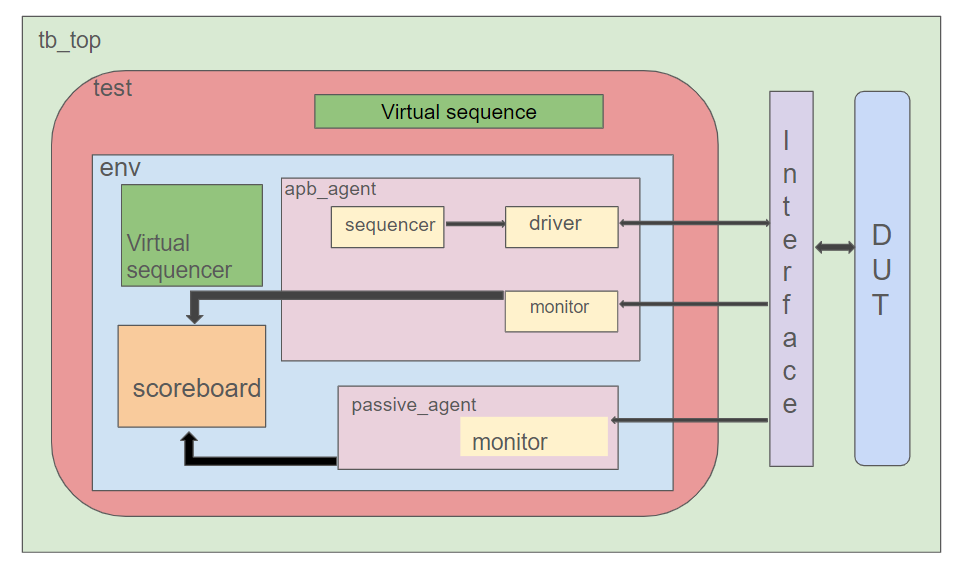
APB Timer tb\_top has the run\_test call which creates the timer UVM test that is passed as argument from the command line. Test instantiates UVM environment. The environment contains active agents which generate and drive stimulus into a given interface and monitor the interface.

**Agents**

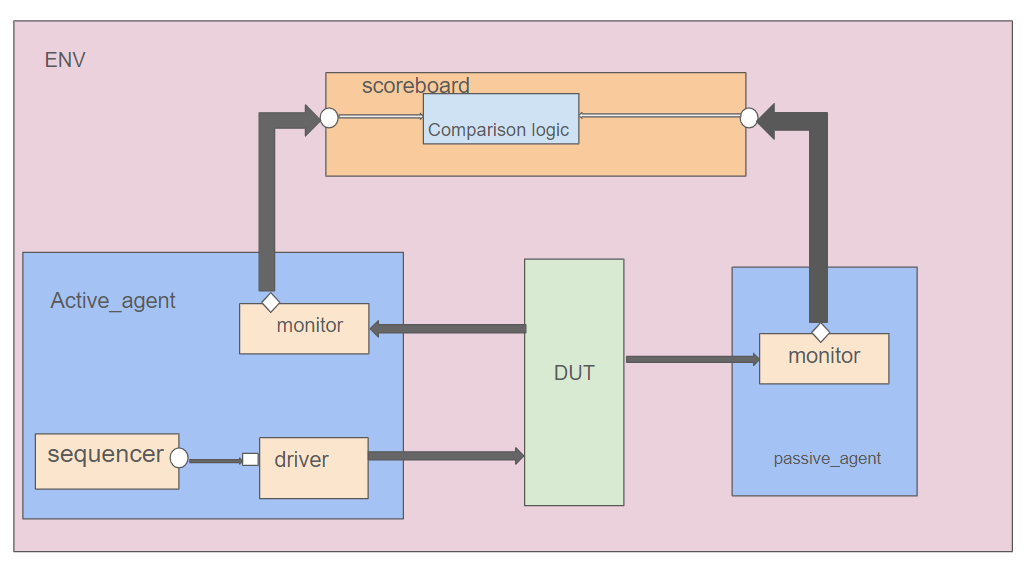
In our project we have one agents namely apb\_Agent. Each agent consists of Driver class, Sequencer class and Monitor Class.

* **Driver** sends data to DUT via virtual interface, before that it accepts the request of sequencer when data is needed, the data is sent through sequencer via TLM and then drives the data to DUT.
* **Sequencer** collects the sequence from sequence class and when request is granted by Driver it sends the collected sequence via TLM.
* **Monitor** collects output data from DUT after the input data through the driver is given.

**Test Bench Architecture:**



**Figure 3: TB Architecture**



**Figure 4: ENV-scoreboard**

**Virtual sequencer and sequence**

Timer environment contains a virtual sequencer.  Virtual sequencer contains handles to all physical sequencers in the environment.  A Timer test can run a virtual sequence on the virtual sequencer.  A virtual sequence would typically call multiple individual sequences to generate stimuli as needed.  As part of the timer test bench, a library of virtual sequences is developed.

**6 Functional Coverage**

**Coverage be done at transaction and signal level.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No.** | **Block** | **Covergroup name** | **Covergroup Type** | **Covergroup Details** | **No.of instance** | **Coverpoint count** | **Coverpoint details** | **Sampled - when/where** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |