|  |  |
| --- | --- |
| Error Correction Encoder & Decoder | **Digital Design and Logical Synthesis for Electric Computer Engineering** |
| **(36113611)**  **Course Project**  **Verification**  **Version 0.1** |

Version 0.1 4 June 2007

**Revision Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rev** | **Change** | **Description** | **Reason for change** | **Done By** | **Date** |
| 0.1 | Initial document |  |  | Shy Hamami | 4,Jun,2007 |
| 0.2 | Digital Changes |  |  | Amir Kolaman | 14,Jul,2007 |
| 0.3 | Functional Verification |  |  | Amir Kolaman | 4,November 2007 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table of Content**

[LISt of figures 5](#_Toc172279519)

[List of tables 6](#_Toc172279520)

[5. Verification Plan 20](#_Toc172279564)

[5.1 Verification Test Objectives 20](#_Toc172279565)

[5.2 Test Bench Architecture and Functionality 20](#_Toc172279566)

[5.2 Test Bench Architecture and Functionality 20](#_Toc172279566)

[5.3 Test Bench Input & Output 20](#_Toc172279567)

[6. Verification Results 21](#_Toc172279568)

[6.1.1 Test <<test name>> 23](#_Toc172279569)

[6.1.2 Test <<test name>> 23](#_Toc172279570)

[6.1.3 Test <<test name>> 23](#_Toc172279571)

[7. Appendix 24](#_Toc172279572)

[7.1 Terminology 24](#_Toc172279573)

[7.2 References 24](#_Toc172279574)

# LISt of figures

[Figure 1: Test Bench High Level Diagram 9](#_Toc181935849)

[Figure 2: Test Bench Block Diagram 9](#_Toc181935849)

# List of tables

[Table 1: Test Plan Functionality 10](#_Toc181935851)

[Table 2: Test Plan FunctionalCheckers 12](#_Toc181935852)

# Verification Plan

Diagram

Description automatically generated

The top module for verification system design is “tb\_overall”, our strategy for defining out test bench start of understanding with what we can check and what we can compare with the use of the seven systems that we learned in Lab 2 to build the verification system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test name/number | Functionality | Test data set | Expected result | Observations |
| Signals\_test | Checking if the amount is right | Rst, PENABLE , PSEL , PWRITE , Opration\_done | We expecting the right amount of high and low signalks | From the observartion the system giving the right amount og high and low |
| Amount\_of\_noise | Checking the amount of noise we had in the test including the Gm | Dut num\_of\_errros,  GM num\_of\_errors | Expecting to get the same amount in both of them and getting 2/3 of errors in total compare to operation\_done | From observartion we got what it need to give, also we getting amont almost the same amount for no\_errors and one and two errors |
| Error\_spot | Cheking that the noise |  |  |  |
| Rst\_active |  |  |  |  |
| RegistersReadCheck |  |  |  |  |
| operation\_done\_active |  |  |  |  |
| ResultCheck |  |  |  |  |
| NumOfErrorsCheck |  |  |  |  |
| NumOfErrorsBoundaryCheck |  |  |  |  |

<<This section should describe the design verification strategy your team has undertaken as a basis for defining your test bench. It should include a list of the operating scenarios you are testing as part of your test bench, and what are the data sets for each test. Preferably, you should include these in a table, listing (1) test name/number, (2) functionality being tested, (3) test data set, (4) expected result, (5) any observations from the actual simulation of the design mode under test. You should provide documentation of the architecture of your test bench, indicating what functions each unit of Verilog code in the test bench carries out. >>

## Verification Test Objectives

The objectives of our testing and verification is full operate module that encode and decode the data that was received.

<<Herein, describe the objectives of your testing and verification of your mode and how you have gone about the process of creating the test bench, i.e., *what* specifically your test bench does. >>

## Test Bench High Level Diagram and Architecture

<<Herein, this section describes the structure of your verification environment. You must explain why you decided to design your test bench that way. What does each unit does (in short) and how they communicate.

Put a diagram of your test bench, you can create it in any tool (HDL Designer, Power Point, Visio etc..). No need to write each signal, but need to describe the general functionally of the test bench. What is the general flow.

Example:

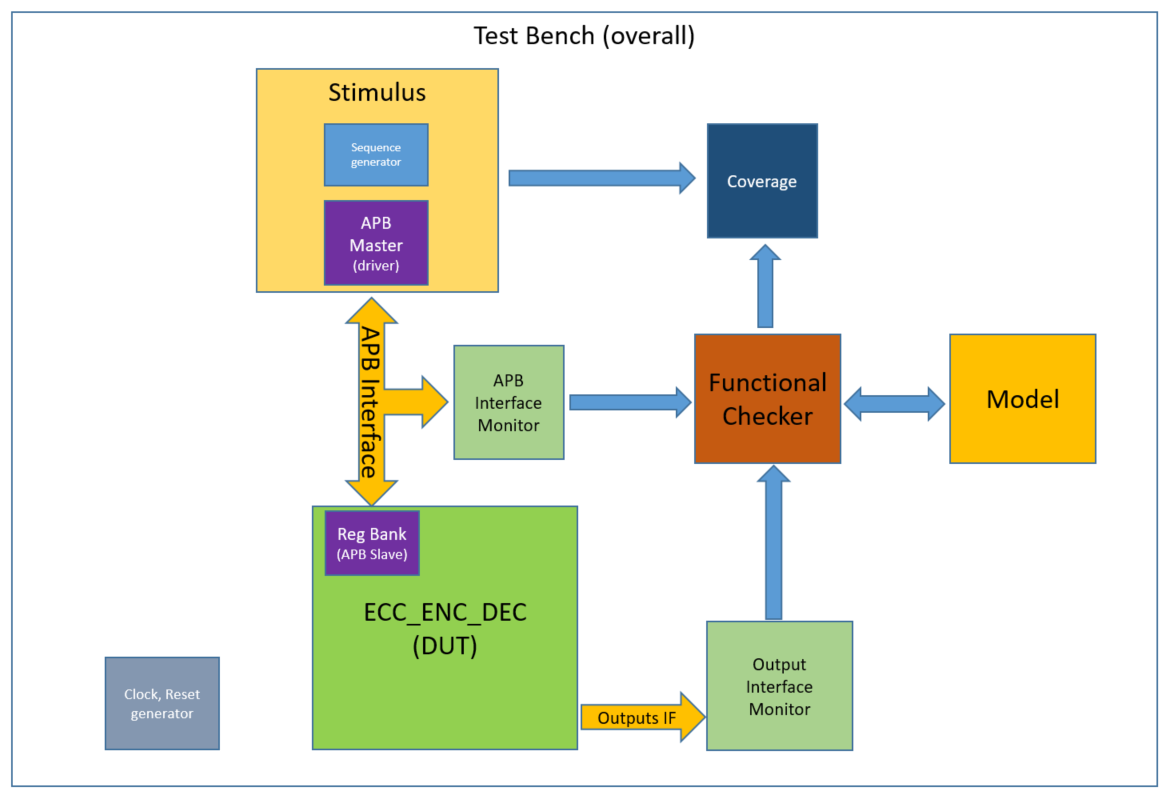


Figure 1: Test Bench High Level Diagram

1. Interface – Contain all the buses between all the new modules and the DUT.
2. Stimulus – In charge of creating random noise and inserting all the needed data form the file into the GoldenModule and the DUT using the AMBA protocol.
3. Coverage - Determines how much functionality of the design has been exercised.   
   Functional assertions are used to check whether each and every corner of the design is explored and functions properly.
4. Checker – Checking that we getting values that are within an acceptable range according to the proper design order , also we checking the input and output properties and compare them with the GoldenModule.
5. Golden Module – We use here the Matlab and the Verilog to send the right output to the checker.

For the verification plan we started by first planning how the structure of our test needs to be, and what are the common signals in our interface.  
The signal for us [add the signal]

The second step is creating the Stimulus – the simulator for this verification. This module is responsible on the design signals, that feeds the whole system – except clk and rst that will be an input for the system, and the output of the DUT.

Next we have the Golden Model, representing a machine that give us perfect outputs for the stimulus comment lines. We did it by making a matlab code that creates random inputs, and their parities. On the other hand, we used a System Verilog module to assert the right answer simultaneously with the DUT.

To check whether golden model’s and the DUT’s output match, we used the Functional checker. This module receives the output of both entities and continuously make sure both are on the same page.

Finally, the Coverage module is designed to monitor the input of our system, checking if the verity of the inputs was satisfying enough, for this verification plan.

The purpose of the stimulus is to .... It contains and APB master which is responsible to toggle APB bus properly. It also contains a sequence generator where we decide what APB transactions are sent to DUT and in what order, the stimulus also contains ……

The decided to connect clock and reset signals from a generator, to all blocks in the testbench, and not from stimulus because….

The role of the APB interface monitor is to track the APB read/write transactions and to translate them to read/write operations, we are sent to checker. The role of the Output Interface Monitor is to ….

We decided to track each interface separately because… The disadvantage of this method is that we assume CPU knows when the DUT finished the previous operation, however the CPU is not connected directly to the ‘operation\_done’ signal, and therefore cannot know when the DUT is ready to perform a new encode/decode operation. We solved that problem by …. However this method also has advantages such as ……

The role of the Checker is to understand what are the inputs to the DUT, what are the outputs from the DUT, and to check if the expected results match the actual results.

We created the APB interface with three modports: dut/slave, stimulus/master, and monitor, because..

The purpose of the Model is to receive input vectors (for example: data\_in to be encoded). It reads from a file that we created with Matlab, and outputs back the output vectors (for example data\_out which is codeword).

The purpose of the Coverage module is to track verification events and signals, and tell us if we covered all the scenarios that needed to be covered.

We debated to which modules we want to connect the Coverage module. First option was to connect it to the two interfaces, so we cover all scenarios directly from signals that are connected to DUT. Second options was to connect Coverage to the Stimulus and Checker so we can easily cover both the transactions that we decide to send to DUT, and the outputs that we get and the scenarios that the Checker found that are happening. The second option has a big disadvantage that we cannot take this testbench to a higher hierarchy (when ECC\_ENC\_DEC is tested as part of a bigger design) by simply disconnecting the stimulus and connecting to another design. The third option was to …… We decided to go with the second option.

## Test Bench Low Level Architecture and Functionality

<<Herein, describe in details the functionality and architecture of your test bench units, i.e., its structure and function, and how it carries out the testing objectives identified in the previous subsection of the document. You might include a block diagram of your test bench units. The functionality of the test bench can be listed in a table according to the module within the test bench architecture.

EXAMPLE:

This counter starts from the loaded value "din" when "load" is asserted. It counts up or down depending on "up\_dn" input. The counter starts counting when "clk\_en" ius asserted. The counter cycles through its values - when it reaches minimum/maximum value, the next count will be maximum/minimum value >>

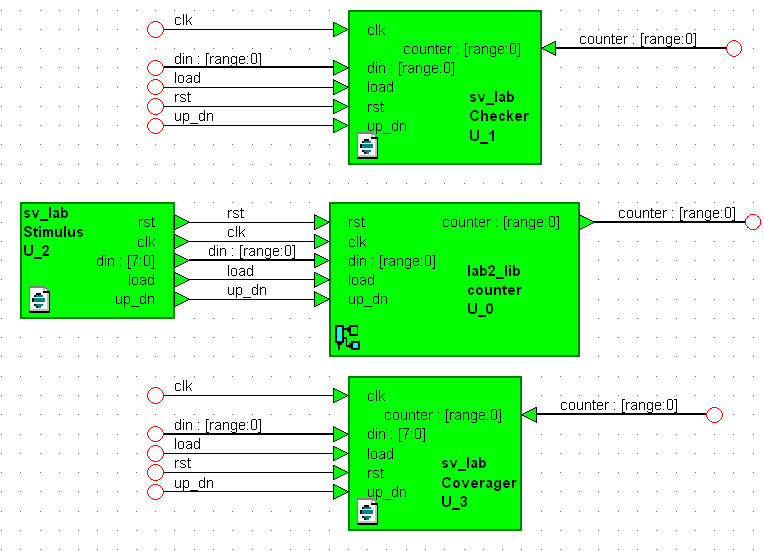


Figure 2: Test Bench Block Diagram

## Functional Coverage

<<Herein, describe the functional coverage for each of the test conditions.

Function – name of the function you want to cover

Event – at what event should the functional coverage be checked?

Coverage point – what is the point to be checked?

Bins – values that should be covered.

Fill in the values in the following table:>>

**<<Instructions:** Divide the Function into two parts: (1) EXTERNAL FUCNTIONALITY – what goes in and what comes out? (2) INTERNATL FUNCTIONALITY – what happens inside

COUNTER EXAMPLE:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FUNCTION** | **EVENT** | **COVERAGE POINT** | **BINS** | **scenario** |
| **Counter Value** | **Posedge clock** | **count** | **0,255,100** | **standard** |
| **Count Up or Down** | **Posedge clock** | **up\_down** | **0,1** | **Standard** |
| **Load** | **Posedge clock** | **Load** | **0,1** | **Standard** |
| **Reset** | **rst** | **rst** | **0,1** | **Standard** |
| **From min to max** | **Posedge clock** | **counter** | **0🡪127** | **Extreme** |
| **From max to min** | **Posedge clock** | **counter** | **127🡪0** | **Extreme** |

1. Test Plan Functional Coverage

.>>

## Test Bench Functional Checkers

<<Herein, describe the functional Checkers for each of the test conditions you have covered, and also describe what are the expected results of the specific test case.

Condition – at what condition should the function be checked?

Expected – what are the expected results?

You can divide the Functional Checkers into two parts:

1. EXTERNAL CHECKERS – what goes in and what comes out?
2. INTERNATL CHECKERS – what happens inside

Fill in the values in the following table:

>>

<<Counter Example external checker:

|  |  |  |
| --- | --- | --- |
| **Condition** | **Expected Result** | **Scenario** |
| **Reset Active in low** | **Data\_out,operation\_done, num\_of\_errors🡪0** | **Standard** |
| **Register Read** | **PRDATA == GM\_RGISTER** | **Standard** |
| **PADDR[3:2] = 00 && PWRITE&&PENABLE && PSEL are on high** | **(Only if we didn’t have Reset)[1-4] Clock poseedge 🡪  operation\_ done = 1** | **Standard** |
| **Operation\_done->1 && num\_of\_errors <= 2** | **Data\_out == GM\_data\_out** | **Standard** |
| **Operation\_done->1** | **Num\_of\_errors == GM\_num\_of\_erros** | **Standard** |
| **Operation\_done->1** | **Num\_of\_errors = [0 : 2]** | **Standard** |

1. Test Plan FunctionalCheckers

## Golden Model

<< Describe in short how your Golden Model works. Do not go into too much details about your high level programming (Python, Matlab etc..) code. If you read data from files, explain in short the structure of this files. >>

# Appendix

## Terminology

**LSB** - Least Significant Bit

**TBR** - To Be Reviewed

**TBD** - To Be Defined

**IF** - Inteface

## References

<<I expect your team’s work to make appropriate citing of relevant literature—whether it is product manuals, the text, or other works you have read or consulted in order to do the project. If you don’t have any references, then leave the section blank. >>