Block NaME

Verification Plan

Author:

Katrin Nekhin

Yuval Saar - 304999261

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Description** | **Done By** | **Date** |
| 1.0 | First Draft | Katrin Nekhin |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Content:

[1. General 4](#_Toc90975912)

[2. Block description 5](#_Toc90975913)

[2.1. DUT High Level Description 5](#_Toc90975914)

[2.2. DUT Block Diagram 5](#_Toc90975915)

[2.3. DUT Interface 6](#_Toc90975916)

[2.4. Special Notes 6](#_Toc90975917)

[3. Testbench 7](#_Toc90975918)

[3.1. Testbench Concept 7](#_Toc90975919)

[3.2. Testbench Diagram 7](#_Toc90975920)

[3.3. TB Blocks detailed description 8](#_Toc90975921)

[3.3.1. Agents 8](#_Toc90975922)

[3.3.1. Sequencers 9](#_Toc90975923)

[3.3.2. Sequences 10](#_Toc90975924)

[3.3.3. Collectors & Monitors 12](#_Toc90975925)

[3.3.4. Scoreboard 12](#_Toc90975926)

[3.3.5. Reference Module 12](#_Toc90975927)

[3.4. Resets & clocks 12](#_Toc90975928)

[3.4.1. Reset 12](#_Toc90975929)

[3.4.2. Clocks 12](#_Toc90975930)

[3.5. Pass / Fail mechanism 12](#_Toc90975931)

[3.6. Start / End test mechanisms 13](#_Toc90975932)

[3.7. Coverage 13](#_Toc90975933)

# General

The purpose of this document is to describe the Verification Plan of the ENC\_DEC\_ECC device, including all of its sub blocks and components, including DUT description, Testbench diagram, detailed mechanism description as well as coverage and checkers tables.

The Verification Plan will especially try to address the following guidelines:

* Allow as much as possible Test / Pattern generation to be re-usable from platform to platform
* Verification will be coverage driven, as to allow smart & fast bring up as well as allow visibility & measurability of the verification status

# Block description

## DUT High Level Description

ECC\_ENC\_DEC is a device with the capabilities of encoding and decoding data.

* Features:
  + Encode Operation
  + Decode Operation
  + Full Channel Operation (encode and decode with added noise)

## DUT Block Diagram

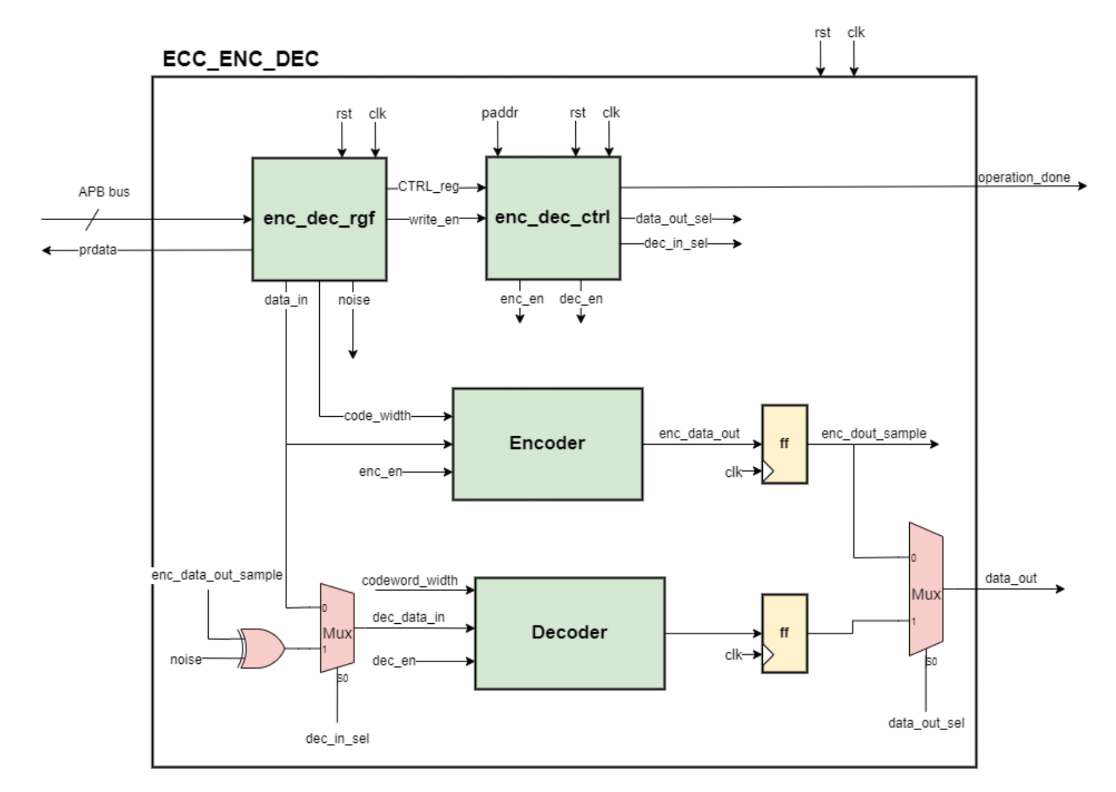


Figure 1 : Block Diagram

## Interface

**Interface Parameters:**

* AMBA\_ADDR\_WIDTH
  + Width of the APB address lines
  + Default value 32
* AMBA\_WORD
  + Width of the data being sent by APB
  + Default value 32
* DATA\_WIDTH
  + Width of a codeword
  + Default value 32

**Interface Signals:**

* operation\_done
  + 1 bit
  + Goes from the DUT to the checker
* data\_out
  + 8/16/32 bit
  + Goes from the DUT to the checker
* Clk, rst
  + 1 bit each
  + Goes into every sub-module
* Paddr, pwdata, penable, psel, pwrite
  + APB input signals, with different sizes
  + Goes from stimulus into the DUT
* Prdata
  + APB output signals, with size AMBA\_WORD
  + Goes from the DUT to the checker
* Num\_of\_errors
  + 2 bit
  + Goes from the DUT to the checker
* Type\_of\_work
  + 2 bit
  + Goes from the stimulus to the checker
  + Indicates what mode is being used (encode, decode, full channel)

# Testbench

## Testbench Concept

The TB handles the following main tasks:

1. Data
   1. For every work mode (Encode/Decode/Full), randomize 10 data inputs and their noise inputs (if in full channel, distributed equally between 0, 1 or 2-bit noise).
2. Configuration
   1. Given every data input, configure the registers via APB.
   2. Send the data input to the golden model.
   3. The DUT handles the input and sends the output to the checker.
3. Checker:
   1. Compare the output with that of the golden model.
   2. Write report.
4. Reset event

## Testbench Diagram

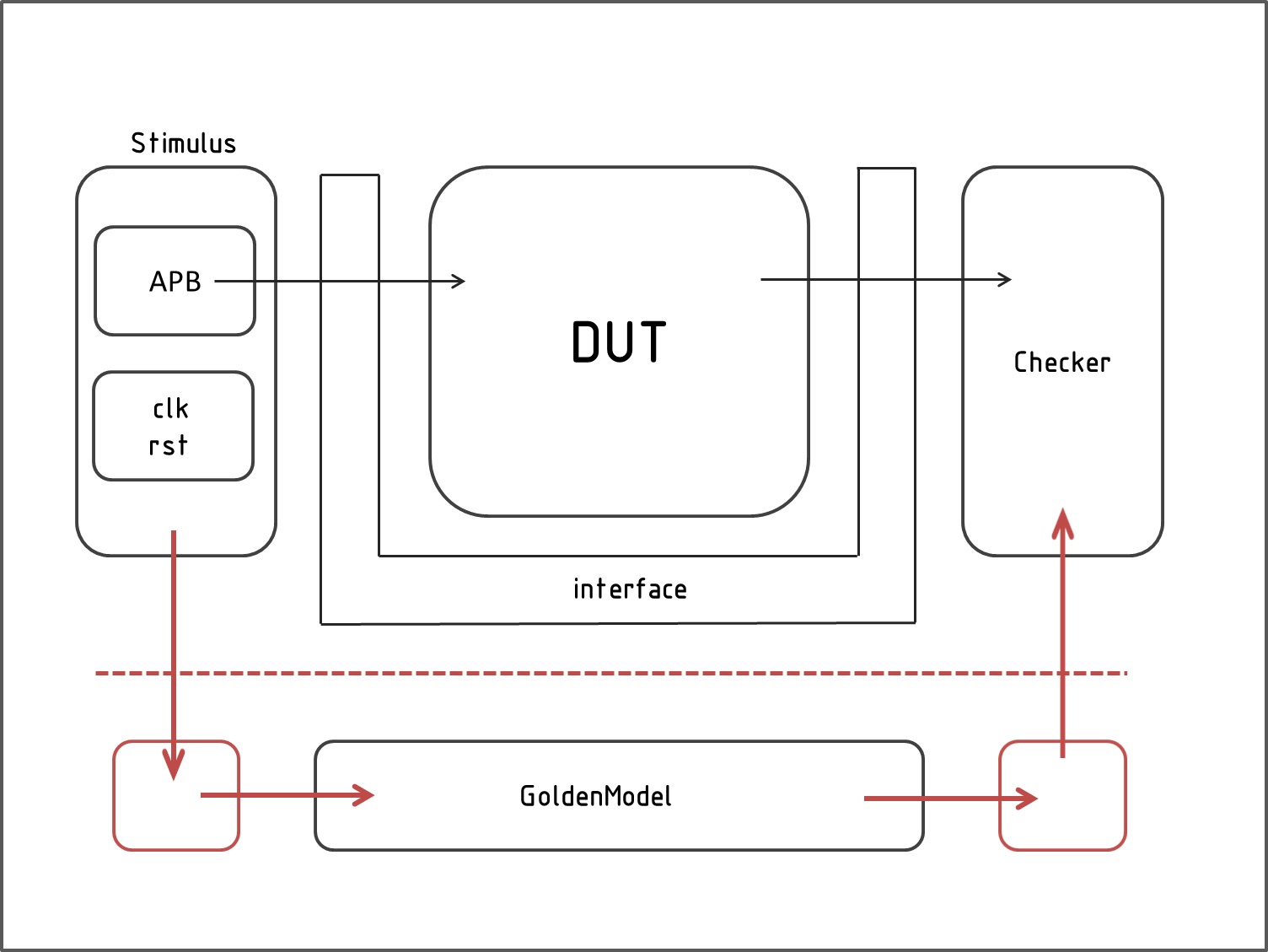


Figure 2: TestBench Active Diagram

## TB Blocks detailed description

### Stimulus

Upon initialization the stimulus generates a list of input data vectors using randomization.

The list is composed of 10 vectors for each mode.

For full channel mode also the noise register vector is randomized between 0, 1 and 2 bit error vectors.

The stimulus then writes the list to a file for the golden model to read, waits for a while for the golden model to finish calculation and then starts sending the vectors to the DUT, one input vector at a time.

Once all of them are sent the stimulus is terminated.

### Checker

The checker waits for the golden model to finish calculations and then reads from the output file, one output at a time.

It then compares the output read to the output received from the DUT and writes the result to a report file.

Checker Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **Event** | **Expected Result** | **Scenario** |
| rst == 1 | rst | data\_out == 0 && num\_of\_errors == 0 | Standard |
| type\_of\_work == 1 && operation\_done == 1 | Encode Operation | data\_out == encoded data\_in | Standard |
| type\_of\_work == 2 && operation\_done == 1 && stim\_num\_of\_errors == 0 | Decode Operation No Noise | data\_out == decoded data\_in && num\_of\_errors == 0 | Standard |
| type\_of\_work == 2 && operation\_done == 1 && stim\_num\_of\_errors == 1 | Decode Operation With 1-bit Noise | data\_out == decoded data\_in && num\_of\_errors == 1 | Standard |
| type\_of\_work == 2 && operation\_done == 1 && stim\_num\_of\_errors == 2 | Decode Operation With 2-bit Noise | num\_of\_errors == 2 | Standard |
| type\_of\_work == 3 && operation\_done == 1 | Full Channel Operation | data\_out == data\_in | Standard |

### Golden Model

The golden model is a MATLAB application which reads data vectors from a text file, performs encoding/decoding on the files according to the operation needed, and writes the results to an output file.

## Coverage

Coverage Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Event | Coverage Point | Bins | Scenario |
| Reset | rst | rst | 0,1 | Standard |
| Control | posedge clock | ctrl | 0:2 | Standard |
| Input data | posedge clock | data\_in | 0,[1:255],[256:65535],[65536:$] | Standard |
| CODEWORD\_WIDTH | posedge clock | codeword\_width | 0:2 | Standard |
| Add noise | posedge clock | add\_noise | 0,1 | Standard |
| Num of errors | posedge clock | err\_num | 0,1,2 | Extreme |

## Resets & clocks

### Reset

* rst\_n
  + Active low “cold\_reset”, synchronized to apb\_clk, resets registers (and datapath?).

### Clocks

* APB clock
  + “always on”
  + Operational clock
  + Register configuration

## Pass / Fail mechanism

Simulation will fail if one of the following events will occur:

* Data does not match expected data – Either by checker compare or by post run script
* Assertions fail

## Start / End test mechanisms

* Start of test will be out of reset
* End of test will be one of:
  + No more transactions
  + Error
  + Timeout by built in TB timeout