UVM for Kmeans IP

[Date]

Liora Huf

Edi Sraiber

Led by: Goel Samuel

Contents

[1 Hardware verification 1](#_Toc52869232)

[1.1 What,why,how? 1](#_Toc52869233)

[2 UVM 2](#_Toc52869234)

[2.1 Structure 2](#_Toc52869235)

[2.2 Coverage 2](#_Toc52869236)

[3 DUT 3](#_Toc52869237)

[3.1 Architectural description 3](#_Toc52869238)

[3.2 Details of parameters that can be varied 3](#_Toc52869239)

[3.3 Description of APB protocol 3](#_Toc52869240)

[4 Implemented Verification environment 4](#_Toc52869241)

[4.1 UVM used classes 4](#_Toc52869242)

[4.2 Ref Model 4](#_Toc52869243)

[5 Test Plan 5](#_Toc52869244)

[5.1 Test Line 1 – Basic Test 5](#_Toc52869245)

[5.2 Test Line 2 – Out as In Test 5](#_Toc52869246)

[5.3 Test Line 2 – Robustness Test 5](#_Toc52869247)

[5.4 6](#_Toc52869248)

[6 Results 7](#_Toc52869249)

[7 Bug Fixes 8](#_Toc52869250)

[8 Conclusion 9](#_Toc52869251)

[9 Bibliography 10](#_Toc52869252)

[10 Appendix A – How to integrate Matlab code to UVM enviroment 11](#_Toc52869253)

# Hardware verification

TBD

## What,why,how?

TBD

# UVM

TBD

## Structure

TBD

## Coverage

TBD

# DUT

TBD

## Architectural description

TBD

## Details of parameters that can be varied

TBD

## Description of APB protocol

TBD

# Implemented Verification environment

TBD

## UVM used classes

TBD

## Ref Model

TBD

# Test Plan

Description of all scenarios to be defined : which parameters will be exercised.

         How is equivalence defined

         Description of Assertions to test APB interface

## Test Line 1 – Basic Test

In this test line, the following parameters will be randomly generated:

1. Number of points
2. Points values
3. Initial Centroid values

These randomly generated parameters should be sent to the DUT and the REF Model. The outputs given by the DUT and the REF Model for the mentioned input shall them be compared, being equivalent if every output centroid presented by the DUT is also presented by the REF Model.

This test line shall produce overall ten tests where each test the randomly generated parameters shall have different values.

## Test Line 2 – Out as In Test

In this test line, the following parameters will be randomly generated:

1. Number of points
2. Points values
3. Initial Centroid values

These randomly generated parameters should be sent to the DUT and the REF Model. After verifying that the outputs given by the DUT and the REF Model for the mentioned input shall are equivalent (every output centroid presented by the DUT is also presented by the REF Model), use the received outputs as input for a new run.

The results expected for the new run are:

1. The output centroids are equal to the input centroids
2. Convergence reached within one iteration

This test line shall produce overall ten tests where each test the randomly generated parameters shall have different values.

## Test Line 3 – Robustness Test

In this test line, the following parameters will be randomly generated:

1. Number of points
2. Points values
3. Initial Centroid values

These randomly generated parameters should be sent to the DUT and the REF Model. The outputs given by the DUT and the REF Model for the mentioned input shall them be compared, being equivalent if every output centroid presented by the DUT is also presented by the REF Model.

This test line will produce multiple (at least 10.000 runs) tests which will be run in series without breaks.

## Test Line 4 – One Iteration Test

In this test line, the following parameters will be randomly generated:

1. Eight data values

These eight values will be used both as points values and initial centroid values. The pass criteria of this test line is to verify that in all runs convergence is reached in one iteration and final centroids are equal to initial centroids.

This test line shall produce overall ten tests where each test the randomly generated parameters shall have different values.

## Test Line 5 – Threshold Test

In this test line, the following parameters will be randomly generated:

1. Number of points
2. Points values
3. Initial Centroid values
4. Convergence threshold value (within a constrain of TBD percent)

These randomly generated parameters should be sent to the DUT and the REF Model. The outputs given by the DUT and the REF Model for the mentioned input shall them be compared, being equivalent if every output centroid presented by the DUT is also presented by the REF Model.

This test line shall produce overall ten tests where each test the randomly generated parameters shall have different values.

## Test Line 6 – Isolated Centroid Test

In this test line, the following parameters will be randomly generated:

1. Number of points
2. Points values
3. Initial Centroid values

Where there one of the following additional constrains:

1. One of the centroids is constrained to be far away from the all the data points. Verify its values does not change (no points are assigned to it)
2. All of the centroids, except from one, are constrained to be far away from the all the data points. Verify their values does not change (no points are assigned to it)

These randomly generated parameters should be sent to the DUT and the REF Model. The outputs given by the DUT and the REF Model for the mentioned input shall them be compared, being equivalent if every output centroid presented by the DUT is also presented by the REF Model.

In addition, in case ‘a’ the isolated centroid value should not change, and no data points shall be assigned to it.

Similarly, in case ‘b’ all data points should be assign to the non-isolated centroid, while the others centroid values shall not change.

This test line shall produce overall ten tests where each test the randomly generated parameters shall have different values.

# Results

TBD

 Results of tests/scenarios run – bug identification

         Results of coverage

         Results of assertions

# Bug Fixes

TBD

## Negative values bug

While building the verification environment, a “sanity check test” done in order to verify if the UVM environments works, the results from the DUT indicated a bug. This bug was apparently connected to the DUT inability to recognize negative values. This bug was fized by the following steps:

* 1. Fix sign representation of variables:

During the calculation, each data point vector to 7 coordinates which shall be represented in fixed point and signed (TBD See reference to chapter blab la in DUT chapters).

* + 1. The variable type of those coordinates were represented in unsigned (default of type in system Verilog is unsigned unless stating "signed" in the type, i.e. signed + type.
    2. The reason for the bug was since it was believed that the compiler will fit to 2's complement when arithmetic operations are being done, yet it did not happened and after diving in a debug process it came up.
    3. The solution was simple in this case and a "signed" syntax was added accordingly for each parsed coordinate process, it shall be noted that as a concatenated vector, the sign does not hold meaning since it matters in coordinate resolution.
    4. The file "accumulator\_adder.sv" changed, as explained above.
  1. Fix 2's complement representation of numbers:
     1. In the summation process of points to form the nominator of the next developed centroids for each iteration, each coordinate holds 22 bits per coordinate(21 + 1 for sign), when each point hold 13(12 + 1 for sign).

See reference to chapter blab la in DUT chapters.

* + 1. When performing arithmetic operations to sum, a negative number represented in 2's complement with 13 bits, wasn’t handled to fit for the operation to be summed to 22 bits number.
    2. The fix was to handled transform the number to its absolute value, then creating the same value in 2's complement representation in 22 bits, then perform arithmetic operations to sum.
    3. The file "distance\_calc.sv" changed, as explained above.

# Conclusion

TBD

# Bibliography

TBD

# Appendix A – How to integrate Matlab code to UVM environment

The following steps should be taken in order to integrate a Matlab function to a System Verilog code. In the case of this report, the Matlab function is used a the Reference Model for the UVM environment.

This is done by exporting a MATLAB function as a component with a direct programming interface (DPI) for use in a System Verilog code.

In order to do so, the following Matlab libraries must be installed:

1. Matlab Coder
2. Matlab HDL Verifier

## DPI Component Generation Steps

1. Write a Matlab function. The some Matlab internal functions are not supported by the DPI generator, therefore after trying to run the generator function, the function code may have to be changed.
2. Write a second matlab function called: build\_dpi. In this function there must be only the *dpigen* Matlab function only.

The *dpigen* receives two mandatory parameters:

* 1. the name of the function intended to be transformed into a DPI component
  2. A flag named *args* followed by the function’s(the function intended to be transformed into a DPI component ) arguments types.

For example, in the case of this project, the Reference Model function receives the following inputs:

1. A 512 by 7 matrix of fixed points numbers, where the integer part is represented by 2 bits, the fractional part is represented by 10 bits and it is signed.
2. A 8 by 7 matrix of fixed points numbers, where the integer part is represented by 2 bits, the fractional part is represented by 10 bits and it is signed.
3. One fixed point number, where the integer part is represented by 2 bits, the fractional part is represented by 10 bits and it is signed.
4. One fixed point number, where the integer part is represented by 13 bits, the fractional part is represented by 0 bits and it not is signed.
5. One fixed point number, where the integer part is represented by 2 bits, the fractional part is represented by 10 bits and it is signed.

Therefore, the depigen command in the case of this projects if the following:

dpigen -args {fi(zeros(512,7),1,13,10,'RoundingMethod','Floor'),fi(zeros(8,7),1,13,10,'RoundingMethod','Floor'),fi(zeros(1,1),1,13,10),fi(zeros(1,1),0,13,0),fi(zeros(1,1),0,13,0)} refModel3.m -rowmajor -launchreport -FixedPointDataType BitVector

Where the named of the function intended to be transformed into a DPI component is *RefModel3.m* .

The used command in this case had additional optional flags for the *dpigen* function, in order to use the Matlab type fixed point type *fi* and how to ”pack” the arguments which are matrixes ( these flags *are -rowmajor -launchreport -FixedPointDataType BitVector*)

For more on the *dpiden* function and optional flags, refer to : <https://www.mathworks.com/help/hdlverifier/ref/dpigen.html>

The *dpigen* function generates a System Verilog DPI component shared library from the chosen MATLAB function and all the functions that the function written in previous steps calls. The generated libraries are:

* 1. .dll for shared libraries if the *build\_dpi* function ir run on Microsoft® Windows® systems
  2. .so for shared libraries on Linux® systems if the *build\_dpi* function ir run on Microsoft® Windows® systems

1. In order to integrate the DPI component in UVM environment, the build\_dpi function must be run on Linux systems(duo to the fact that a .so file is needed). Therefore, created a Matlab folder within the project files folder in a Linux system containing the function intended to be converted to DPI component and the *build\_dpi* function.
2. Run the *build\_dpi* function. The function will create the needed libraries and .sv files within the path *Matlab\_folder*/codegen/so/*function\_name*, where the *Matlab\_folder* is the named of the folder created in the previous step and the *function\_name* is the name of the function intended to be converted to DPI component.
3. In the UVM TBD file, include the dpi generated files:

Include *Matlab\_folder*/codegen/so/*function\_name\_dpi.sv*

Include *Matlab\_folder*/codegen/so/*function\_name\_dpi\_pkg.sv*