An MPEG-4 DECODER DESIGN VERIFICATION USING FUNCTIONAL COVERAGE

Leandro M. de L. Silva, Romulo C. P. Camara, Maria de L. N. Neta, Helder F. de A. Oliveira,
Fabrício G. L. de Melo, Karina R. G. da Silva and Elmar U. K. Melcher
{leandro, romulo, maria, helder, fabricio, karina, elmar}@lad.dsc.ufcg.edu.br
Universidade Federal de Campina Grande
Aprígio Veloso Avenue, 882, Bodocongó
Campina Grande - PB - Brasil

ABSTRACT

Functional coverage is a simulation progress measurement. Using functional coverage is possible to know if all the functionalities have been tested and the simulation can be finished. The proposal of this work is to present a case study of coverage applied to some blocks from MPEG-4 decoder design.

1. INTRODUCTION

Functional verification can be used to verify if the implemented design is in accordance with the specification [1]. One of the biggest problems in order to implement functional verification is to assure that all specified functionalities have been exercised.

To solve this problem, coverage is being used. Functional coverage measures the progress of the simulation and report, which functionalities have not been exercised or have been exercised more than once. It can also help to inspect the verification quality and to bias the generation of stimuli in order to catch uncovered functionalities, which are usually called coverage holes.

Some works have been produced aimed at improving coverage. In [2] a method for defining views onto the coverage data of cross-product functional coverage models is proposed based on Selections, Projections and Groupings in order to find the coverage hole. In other work [3] techniques to discover and report coverage holes are shown, based on the analysis of the Coverage Model. It uses the definition of aggregated holes, partitioned holes and projected holes and shows the uncovered and lightly covered areas in cross-product functional coverage models. Finally, work [4] defines the coverage methodology, which proposes the separation of the coverage model definition from the coverage analysis tool.

The objective of this work is to show a verification application using simulation, random stimuli and functional coverage in an MPEG-4 decoder design, in order to show if all specified functionalities are exercised during the simulation and find errors as soon as possible in the design. In section 2 we will show VeriSC traditional, the Verification Methodology used in this work, and in section 3 we will describe the MPEG-4 decoder design and the ACDCIP (AC and DC Prediction Inverse), IQ (Inverse Quatisation), RGB(Red Green Blue Block), PBC(Prediction Block Copy) and IDCT (Inverse Discrete Cosine Transform) modules of the MPEG-4 decoder which were verified using the coverage process.

The remaining of the sections are organized as follows: Section 4 shows the development methodology, Section 5 states some results and Section 6 shows the conclusion.

2. FUNCTIONAL VERIFICATION

The most difficult challenge in the design of any system is to make sure that the final implementation is free of implementation flaws. The traditional VeriSC methodology automatically creates a DUV-specific template testbench. It creates a SystemC based object oriented environment to perform verification [5].

In traditional VeriSC methodology the testbench is composed of the following elements: Source, TDriver(s), TMonitor(s), Reference Model (RM) and Checker. The testbench function is to input data into the DUV and the RM, capturing the outputs and comparing automatically if both of them are equivalent. The synchronization mechanism of the testbench is implemented by means of First-In-First-Out queues (FIFOs), through which transactions are sent. The testbench structure is intended to be used only for synchronous designs with a single clock signal.

Together with VeriSC, BVE_COVER library has been used in order to implement functional coverage and find coverage holes.

3. MPEG-4 DECODER DESIGN

The test case is a part of an MPEG-4 video decoder IP core which is an OCP-IP compliant open-source SystemC-RTL. It came from an effort of Brazilian government who created some programs with the objective of create some human resources in the microelectronics area. One of them is the Brazil-IP [6], in which the MPEG-4 decoder design has been conceived. Today there is a first-time silicon chip [7]. The chip is ready and the layout has 22.7mm² at a 0.35µm CMOS 4ML technology with a 37MHz working frequency. The schematic block from the MPEG-4 decoder can be seen in Figure 1.

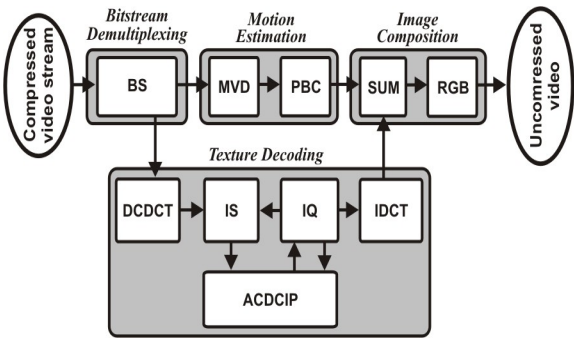

Figure 1: The MPEG-4 schematic block

The MPEG-4 decoder design has the characteristics that can be seen in Table 1.

Golden Model code lines	12 718
RTL code lines	21 473
Testbench code lines	51 382
Test vectors	50 789 096

Table 1: MPEG-4 characteristics

The modules from the MPEG-4 used in this paper are the following: ACDCIP, IQ, RGB and PBC.

The ACDCIP is the module responsible for predicting the next coefficient blocks based on the previous ones by means of an inverse prediction function. More precisely, it is necessary the information about the DC coefficients of the previous dequantized blocks to calculate the prediction direction and the prediction values [8].

The IQ module is characterized by dequantize the coefficient blocks by means of a multiplication of these coefficients by a scaling factor, so a coefficients array is dequantized to produce the reconstructed DCT coefficients. The inverse quantization process has two methods:

1. The first method is used when the variable quant_type is equals 1. All coefficients other than the DC coefficient of intra block needing a two weighting matrices are used. One is used for intra macroblocks and the other for non-intra macroblocks.
2. The second method is used when quant_type == 0. The inverse quantization of DC coefficient is quantized using the same method as in the first inverse quantization method. The change occurs in the quantizer_scale and the stepsize, it will be twice the quantizer_scale.

The RGB module is module responsible for calculating the R, G and B values from Y, Cr and Cb values of the pixels (x,y). RGB are respectively the red, green and blue values of a frame. The Y pixel is the Luma value, Cr and Cb are the Chroma values. The variance interval of RGB is 0 to 255, where 0 is the darkest color and 255 is the lightest color. Cr and Cb are values received of the SUM module and Y is the values, which came of the actual memory images output. There are two variables, x and y, that are values of coordinates x and y of the pixel in the image. These values do not use all values of 8 bits. The variable x uses an interval of 0 to 175 and y uses an interval of 0 to 143.

The IDCT module is the module responsible for transforming the coefficients matrix in a samples matrix. Mathematically, the IDCT operates on Y, an NxN matrix of coefficients, and creates X, an NxN matrix of samples. The action is described in terms of a transform matrix A. The forward IDCT of an NxN coefficients matrix is defined as:

$$X = A^T Y A$$

Equation 1: Representation of IDCT

The elements are:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N}$$, where

$$C_i = \sqrt{\frac{1}{N}}(i=0), \text{ or } C_i = \sqrt{\frac{2}{N}}(i>0)$$

Equation 2: Transform matrix

The PBC module is the module responsible for reading from previous frame values of luminance (represented by Y) and chrominance (represented by U and V) to compose the next image, for each macroblock. The luminance and chrominance receive unsigned values with 8 bits, which give the interval from 0 to 255. Motion vectors (MVxy) are entry dates of PBC module too. They are received from the MVD module. The MVxy from each block of luminance inform where are the luminance values from previous frame. MVxy for chrominance values should be calculated with the average of luminance MVxy. The variance interval of Motion Vector values is -64 to 63.

4. METHODOLOGY

The MPEG-4 decoder design has been verified using the VeriSC methodology. More details can be seen in [5].

The followed steps to verify the MPEG-4 decoder design are specified following.

1. The verification environment (testbench) used a Reference Model as a Golden Model to the MPEG-4 decoder. The Reference Model is the XVID software [9].
2. The second step was to implement the hierarchical division from MPEG-4 decoder design.
3. Each module from MPEG-4 decoder was implemented and verified for different engineers.
4. The verification was implemented using simulation and the results coming from RTL was compared with the results coming from the Reference Model.
5. Coverage was used to show if all specified functionalities has been exercised. In order to do this, coverage plans were created for each module to specify the aims of the verification. Following, each group implemented the coverage for each module, complying with the coverage plans.
6. The results from coverage were analyzed and the stimuli were directed based on these results. Figure 2 shows the process used during the simulation.

In order to implement functional coverage, a SystemC Coverage library was used. It is called BVE_COVER Library. The BVE-COVER library is used to implement coverage-driven simulation. It is composed of 4 basic components:

- **BVE_COVER Bucket**: contains all the functionalities that must be exercised, as well as the number of times that desires to cover them.

- **BVE_COVER Ignore**: they are functionalities of the Bucket that are not important for the simulation and must be placed in the specification in order to show false coverage holes.
- **BVE_COVER Illegal**: it must be placed in the functionalities that should be not executed during simulation. It plays a role similar to BVE_COVER Ignore, but it works like an assertion showing errors.
- **BVE_COVER Cross_coverage:** it crosses buckets to show if the two functionalities of buckets are executed at the same time.

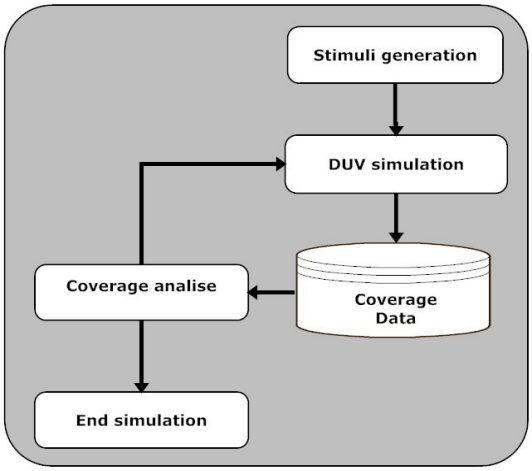

Figure 2: stimuli generation flow

Random-constrained stimuli were used to verify the Design Under Verification (DUV).

Based in the specified functionalities that the BVE_COVER coverage library contains, a progress bar shows the progress of the simulation, as shown in Figure 3. Each functionality bucket is shown with a percentage. Then, is possible to observe the verification progress.

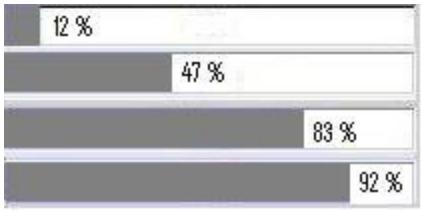

Figure 3: Simulation progress from BVE_COVER library

5. RESULTS

The functional coverage of the MPEG-4 decoder design was implemented in two stages. In the first stage a weak coverage test was implemented, where a few functionalities were tested, and in the second stage the coverage was improved, adding some coverage on the new functionalities. This improvement has resulted in one error detection before the back-end flow (the layout phase) step.

This error detection was the most relevant result of the application of functional coverage in the MPEG-4 decoder design. It was a functional error detected in the ACDCIP module.

The error found consisted of a difference between the values received by the *Checker* from the *Reference Model* and the *DUV* of that module. It was found comparing the values received in the 86º test vector at approximately 21 minutes of the simulation, when the coverage process has reached 37% of completeness. After that, the error in the PIACDC module was fixed and the MPEG-4 decoder chip could be succefully conceived.

Some other results were obtained from the simulation and verification process, as can be seen in Table 2. It shows the specified functionalities in each module and the reached coverage respectively.

	IDCT	ACDCIP/QI	RGB	PBC
Functionalities number	52	14	58	33
Coverage (%)	100	100	100	100

Table 2: Specified functionalities x coverage

6. CONCLUSIONS

This paper presented a functional coverage approach applied to an MPEG-4 decoder design, specifically in the modules ACDCIP, IQ, RGB, PBC and IDCT. The experiments detected the occurrence of one functional error in the ACDCIP module and a speed up in the process of verification for some modules of the MPEG-4. This result was very important because it provided an economy of time and money in the development process, making capable the production of a first-time MPEG-4 silicon chip.

7. ACKNOWLEDGMENTS

To the National Counsel of Technological and Scientific Development–CNPq by the scientific initiation resources that make possible this work takes place, to our advisor by the advices and constant revisions and to the work partners by the help, patience and friendship all the time.

8. REFERENCES

[1] J. Bergeron, editor. Writing Testbenches. Springer, Boston, 2003.

[2] S. Asaf, E. Marcus, and A. Ziv. Defining coverage views to improve functional coverage analysis. In DAC ’04, pages 41–44, New York, 2004.

[3] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole analysis for functional coverage data, 2002.

[4] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage: a tool supported methodology for design verification. In DAC ’98, pages 158–163, New York, 1998.

[5] Karina R. G. da Silva, Elmar U. K. Melcher, Guido Araujo, and Valdiney Alves Pimenta. An automatic testbench generation tool for a systemc functional verification methodology. In SBCCI ’04: Pro-

ceedings of the 17th symposium on Integrated circuits and system design, pages 66–70, New York, NY, USA, 2004. ACM Press.
[6] BrazilIp. www.brazilip.org.br/fenix. 2007.
[7] K. R. G. da Silva, E. U. K. Melcher, I. Maia, and H. do N. Cunha. A methodology aimed at better integration of functional verification and rtl design. Design Automation for Embedded Systems, Volume 10, Number 4 / December, 2005, pp. 285-298.
[8] I. 14496-2:2001(E). Coding of Audio-Visual Objects - Part 2: Visual, Dec 2001.
[9] XVID Software. www.xvid.org. 2006