Video Electronics Standards Association

**39899 Balentine Drive, Suite 125**
**Newark, CA 94560**

**Phone: 510 651 5122**
**Fax: 510 651 5127**
**URL: www.vesa.org**

# VESA Display Stream Compression
## Conformance Test Guideline (DSC CTG)

**Version 1.0**

**27 April, 2015**

### Purpose

The purpose of this document is to provide a test guideline for the VESA® Display Stream Compression (DSC) Standard that can be used by adopters to incorporate into their own test plans.

### Summary

The Display Stream Configuration Conformance Test Guideline (DSC CTG; *DSC CTG v1.0*) is a recommended test flow and accompanying software for testing encoder or decoder implementations of the VESA DSC Standard (*DSC v1.1*). The package contains this document, software scripts, associated files, and recommended pictures.

This testing recommendation is Transport layer-independent and should be incorporated into a test suite that comprehends transport of bits from a Source device to a Sink device.

# Contents

# Tables

# Figures

# Preface

## Intellectual Property



While every precaution has been taken in the preparation of this Guideline, the Video Electronics Standards Association and its contributors assume no responsibility for errors or omissions and make no warranties, expressed or implied, of functionality or suitability for any purpose.

## Trademarks

DisplayPort is a trademark and VESA is a registered trademark of the Video Electronics Standards Association.

All other trademarks used within this document are the property of their respective owners.

## Support for this Guideline

To obtain the latest Guideline and any support documentation, contact VESA.

If you have a product that incorporates Display Stream Compression (DSC), ask the company that manufactured your product for assistance. If you are a manufacturer, VESA can assist you with any clarification you require. Submit all comments or reported errors to VESA, in writing, using one of the following methods:

**Fax:** 510 651 5127, direct this fax to VESA Technical Support

**Email:** support@vesa.org

**Mail:** Technical Support
Video Electronics Standards Association
39899 Balentine Drive, Suite 125
Newark, CA 94560

## Acknowledgements

This document would not have been possible without the efforts of VESA's DSC CTG Task Group. In particular, Table 1 lists the individuals and their companies that contributed significant time and knowledge to this version of the Guideline.

**Table 1: Main Contributors to *DSC CTG v1.0***

| Name | Company | Contribution |
|---|---|---|
| Jim Hunkins | Advanced Micro Devices, Inc. | Author, Co-chair DSC Task Group |
| Fred Walls | Broadcom Corporation | Contributor |
| Simon Bussières | Hardent | Contributor |
| David Stears | NVIDIA Corporation | Contributor |
| James Goel | Qualcomm, Inc. | Author |
| Harris Chan | Qualcomm, Inc. | Contributor |
| Dale Stolitzka | Samsung Display Co., Ltd. | Chair DSC Task Group, Reviewer |

## Revision History

**Table 2: Revision History**

| Date | Version | Description |
|---|---|---|
| April 27, 2015 | 1.0 | Initial release of the Guideline. |

# 1 Introduction

## 1.1 Scope

The intended audience of this guideline includes encoder and decoder designers, implementers, transport specification writers, and test coverage engineers.

This document applies to the DSC functions and related bitstreams. The Transport layer that conveys the DSC bitstream between the encoder and decoder implementations is outside the scope of this document. It is assumed that testing for the Transport layer is covered elsewhere.

Some modes of the VESA DSC Standard (*DSC v1.1*) are not commonly implemented, and therefore are not directly covered within the scope of this document. For example, all testing in this document assumes a Constant Bit Rate (CBR) mode and does ***not*** apply to DSC's Variable Bit Rate (VBR) mode.

This version of the CTG is focused on testing RGB 4:4:4 at this time. The included scripts and sample files are also set for 8 bits per component (bpc). Modifications to such scripts and files can be made to support other color bit depth along with other DSC-supported color spaces. See the DSC Software Model in *DSC v1.1* for information regarding how to modify scripts and files to support different configureations/modes. DSC Software Model toolset updates and release notes, as well as the accompanying CTG fileset updates, should be checked as they become available.

## 1.2 Document Organization

This document is organized as follows:

- Section 1 – Introduction

  This section provides a basic explanation of what this document covers and its expected readership. This section also includes a glossary of terms and references used within the document.

- Section 2 – Overview and Section 3 – CTG System Conformance Validation Model

  These sections describe a workflow that uses pictures and a method for constructing pictures to verify operation of an implementation under test.

- Section 4 – Tests

  This section describes picture-specific tests using the DSC Software Model to cover different aspects of the encoder and decoder implementations.

- Section 5 – DSC CTG Buffer Conformance Tool

  This section describes the test scripts and methods that can expand test coverage.

- Appendix A – Installing Software Required for Tools

  This appendix provides an installation guide for the software required by the tests.

- Appendix B – Usage Notes on ImageMagick for Picture Sizing

  This appendix provides notes for using ImageMagick to properly resize pictures for use with these tools.

- Appendix C – CTG File Summary

  This appendix summarizes the files and formats used by the testing tool.

- Appendix D – DSC Encoder/Decoder Picture Coverage Information

  This appendix summarizes the coverages by each suggested picture.

- Appendix E – Test Pictures

  This appendix provides information related to all included test pictures.

- Appendix F – Use of .cfg Files with Test Pictures

  This appendix includes a sample .cfg file and information related to using the file with different package tests.

## 1.3 Acronyms, Abbreviations, and Initialisms

**Table 1-1: Acronyms, Abbreviations, and Initialisms**

| Acronym/Abbreviation/ Initialism | Stands for |
|---|---|
| BP | Block Prediction. |
| bpc | bits per component. |
| bpp | bits per pixel. |
| CBR | Constant Bit Rate. |
| .cfg | DSC Software Model's configuration filename extension. |
| .crc | cyclical redundancy check filename extension. |
| CRC | Cyclical Redundancy Check. |
| CRC-32 | 32-bit CRC method, as defined in ITU-T Recommendation V.42. |
| .csv | comma separated value filename extension. |
| CTG | Conformance Test Guideline. |
| .dpx | SMPTE S268 filename extension. |
| .dsc | display stream compression filename extension. |
| DSC | Display Stream Compression (VESA). |
| ICH | Indexed Color History. |
| MPP | MidPoint Prediction. |
| .png | portable network graphics filename extension. |
| .ppm | portable pixel map filename extension. |
| PPS | Picture Parameter Set. |
| .py | python script filename extension. |
| QP | Quantization Parameter. |
| .raw | Filename extension/file format that contains only raw pixel data without header information. All parameters of the picture contained in .raw format need to be known to decode the file data. |
| RC | Rate Control. |
| VBR | Variable Bit Rate. |
| VESA | Video Electronics Standards Association. |
| VLC | Variable Length Code. |
| .xlsm | Microsoft Excel 2007 macro-enabled spreadsheet filename extension. |

## 1.4 Glossary

**Table 1-2: Glossary of Terms**

| Term | Definition |
|---|---|
| 4:2:2 | Format for YCbCr video in which the chrominance components are horizontally subsampled by 2. |
| 4:4:4 | Format for RGB or YCbCr video in which the chrominance components are not subsampled. |
| Bits per component | bpc. Number of bits for each of R, G, and B, or Y, Cb, and Cr in the source format of the encoder or destination format of the decoder. |
| Bits per pixel | bpp. Number of bits sent from an encoder and received by a decoder per unit of pixel time. The bits per pixel rate can have a non-integer value, in which case the number of bits received averaged over a number of successive pixels is an integer. |
| Bitstream | Compressed pixel data stream of bits conforming to *DSC v1.1*. Represents the effects of the multiplexing functions specified by this Guideline, as well as the various layers. See Layer. |
| Buffer_stress | Test type designed to stress buffer flow controls that prevent overflow and underflow of the encoder/decoder buffer. |
| Checksum | Numerically calculated value in 32 bits that verifies a bitstream, uncompressed source picture, or reconstructed picture. |
| Decoder | Implementation of *DSC v1.1* that reads a bitstream and generates a reconstructed picture. |
| Encoder | Implementation of *DSC v1.1* that reads an uncompressed pixel stream or data file and generates a bitstream. See Bitstream. |
| Layer | Portion of the hierarchy used in *DSC v1.1*. A DSC bitstream can differ from a combination of bits from different layers due to the actions of the multiplexing functions specified within the Standard. See Bitstream. |
| Picture Parameter Set | PPS. Set of parameters that is optionally transmitted at the start of a coded picture, which provides information necessary to decode the picture. |
| Python | Computer scripting language used in the DSC CTG Toolset. (See Section 5.) |
| Reconstructed picture | Picture created as the output of the DSC decoder. |
| Sink device | Functional block that contains at least one decoder implementation of *DSC v1.1* and a reconstructed pixel stream output. |
| Slice | Independently decodable set of compressed bits that represents a specified set of samples. The set of samples forms a rectangle in the horizontal and vertical dimensions. Decoding of any one slice does not depend on the availability of another slice or on the decoded result of another slice. |
| Source device | Functional block that contains at least one encoder implementation of *DSC v1.1* and a picture source or uncompressed pixel stream to an encoder. |
| Uncompressed source picture | Picture used as the input to the DSC encoder. |
| Visually lossless | Difference between an uncompressed source picture or picture sequence and the same picture or picture sequence after compression and reconstruction that is not detectable to the eye. |

## 1.5 Reference Documents

**Table 1-3: Reference Documents**

| Document | Version/ Revision | Referenced As | Date |
|---|---|---|---|
| *CCITT-32* (ITU-T Reference to the standardized 32-bit checksum algorithm. (CRC-16 is ITU-T Rec. X.25) | Version 0.42 | | March 2002 |
| VESA Glossary of Terms – see www.vesa.org | Current | | Current |
| *VESA DSC C Model* – see www.vesa.org | Version 1.31 | | July 15, 2014 |
| *VESA Display Stream Compression (DSC) Standard* – see www.vesa.org | Version 1.1 | *DSC v1.1* | August 1, 2014 |
| *VESA Intellectual Property Rights (IPR) Policy* – see www.vesa.org/wp-content/uploads/2014/04/VP-200C.pdf | 200C | | April 2014 |

Additional published notes may be made available at future times for additional guidance in using the DSC Software Model, the DSC CTG, and/or different display standard transports related to DSC. It is up to the reader to check the appropriate locations for such additional documentation.

# 2 Overview

This guideline is a package of documentation, pictures, and scripts intended to assist testing a VESA DSC encoder or decoder to determine compliance of an implementation with *DSC v1.1*. *DSC v1.1* specifies a coding system and a Picture Parameter Set (PPS) that conveys a set of parameters that is necessary to encode a picture and decode the compressed picture. Specific implementations may be tested within the context of a specific Transport layer using the guidelines and tools provided within this package. *DSC v1.1* specifies both an encoding and decoding process that exactly determines the output of the signal chain stages illustrated in Figure 2-1. Stage 2 encodes a given picture represented as an uncompressed pixel stream, using the normative C code provided in *DSC v1.1* with a given PPS. The output of the encoding yields an identical bitstream at the output of Stage 2. Likewise, Stage 5 decodes any bitstream into an identical reconstructed pixel stream in Stage 6. The deterministic nature of *DSC v1.1* permits the identical comparison of the observable bitstream transported between Stage 3 and Stage 4 with the encoded representations documented in this package and the bit-exact comparison of the reconstructed pixel stream with decoded picture representations documented in this package.

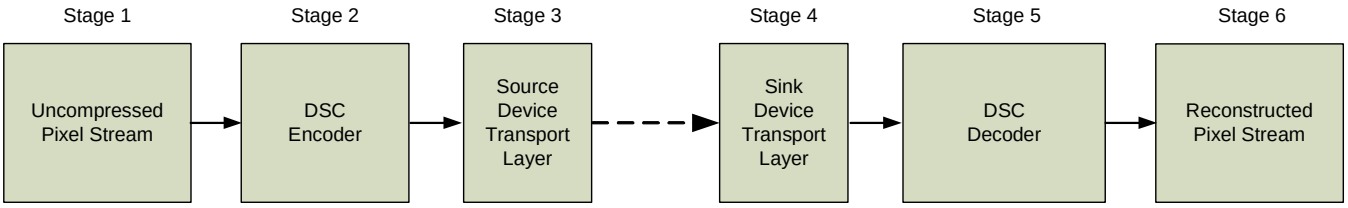


**Figure 2-1: Typical Display Data Flow**

## 2.1 Stage 1 – Uncompressed Pixel Stream

The uncompressed pixel stream represents any possible picture to be encoded by the DSC encoder. The DSC executable program, DSC.exe, reads both .dpx (SMPTE S268) and .ppm (portable pixel map) file formats. When using YCbCr modes (either 4:4:4 or 4:2:2), DSC.exe supports only the .dpx file format.

## 2.2 Stage 2 – VESA DSC Encoder

The encoder compresses the full pixel display into a fixed rate compressed bitstream.

## 2.3 Stage 3 – Source Device Transport Layer

Stage 3 includes all components and protocols that allow a Source device to transport the bitstream from a DSC encoder. This portion of the Transport layer is outside the scope of this document. *DSC v1.1* includes an output rate buffer to convert the variable rate of the entropy encoder to the fixed rate of the link. This rate buffer cannot overflow or underflow.

## 2.4 Stage 4 – Sink Device Transport Layer

Stage 4 includes all components and protocols that allow a Sink device to receive a bitstream into a DSC decoder. All portions of the Transport layer are outside the scope of this document. *DSC v1.1* includes an input rate buffer as part of the Standard to convert the fixed rate of the link to the variable rate required by the entropy decoders. This rate buffer cannot overflow or underflow.

## 2.5 Stage 5 – VESA DSC Decoder

The VESA DSC decoder decodes the compressed bitstream, resulting in the reconstructed picture. This reconstructed picture should be identical every time that the same compressed bitstream is decoded.

## 2.6 Stage 6 – Reconstructed Pixel Stream

The picture-receiving unit may display the reconstructed pixel stream. The DSC.exe program will create a .dpx or .ppm file format.

# 3 CTG System Conformance Validation Model

Interoperability testing of the mobile display end-to-end system is performed in simulation using the VESA DSC Compression and Reconstruction Software Model. Figure 3-1 illustrates a typical system conformance validation flow using cyclic redundancy checks (CRC) at critical stages. This same process is followed regardless of the different configurations suggested in the CTG.

This guideline package uses the CCITT-32 cyclical redundancy calculation value as an efficient way to compare the uncompressed pixel stream, bitstream, and reconstructed pixel stream from *DSC v1.1* with a device under test. A script using this method is included in the CTG Buffer Conformance Package. If using a hardware-generated CRC that is different, it is the tester's responsibility to modify the supplied CRC script to match, as needed. The CRC should be run only on the raw pixel data (uncompressed, compressed, or reconstructed), excluding any file headers or transport-added additional structures.

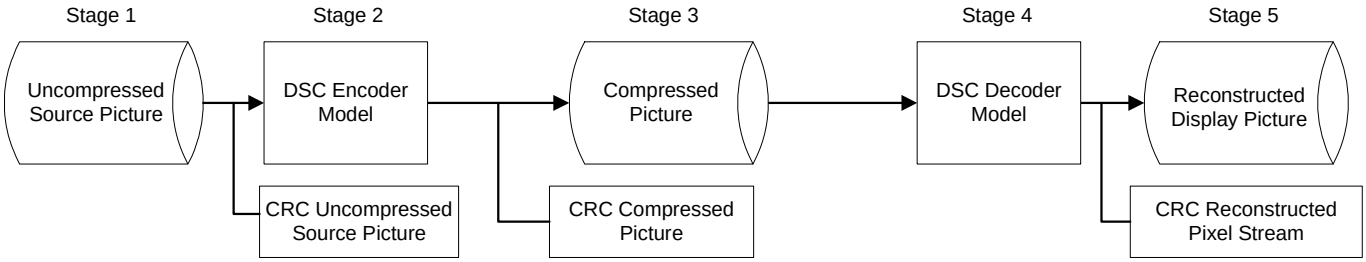


**Figure 3-1: Conformance Guideline Validation Flow**

## 3.1 Stage 1 – Uncompressed Source Picture

This source picture file is uncompressed display data that should match the intended final display resolution and other parameters. Multiple passes through the validation flow are used with different source picture files. The contents of these picture files should be selected to stress the encoder/ decoder functional paths and the rate buffer overflow and underflow limits.

Some transport implementations may have further transport buffer considerations in addition to the DSC buffers. In those cases, the pictures could be designed to stress both the DSC and transport buffers working together.

## 3.2 Stage 2 – DSC Encoder Model

The encoder can be tested independently with the source picture data files and the related CRC for the uncompressed source picture and the compressed bitstream

The uncompressed source picture file from Stage 1 is compressed using a PPS defined to match the required transport rate of the Stage 3 transport model. The VESA DSC encoder executable model is used as the reference for this portion of the digital simulation so each compressed pixel is compared and validated. The compressed output bitstream should match that of the DSC encoder executable model.

Each time the same source picture is run through the encoder, the encoder should always produce the same CRC at the end of Stage 2.

## 3.3 Stage 3 – Compressed Data Transport Layer

The compressed bitstream produced in Stage 2 should be transported through the desired display standard's Transport layer. This test model assumes that no data corruption will occur during the transport. The input data to Stage 3 should exactly match the Transport layer's output data. Any transport-added pieces need to be removed prior to submitting to the DSC Decoder Model. The CRC value will not change as the compressed data passes through Stage 3.

## 3.4 Stage 4 – DSC Decoder Model

The decoder can be tested in the same manner but with the compressed bitstream, the CRC for the compressed bitstream, and the CRC for the reconstructed pixel stream.

The VESA DSC decoder model is used at this stage to reconstruct the pixel data and provide a visually lossless picture. The reconstructed pixel stream from the decoder under test should meet that of the VESA DSC decoder executable model.

## 3.5 Stage 5 – Reconstructed Picture Display

As expected, the original uncompressed picture will not match bit-for-bit to the reconstructed picture because the picture has been lossy compressed.

If the same compressed bitstream is run through more than once, the reconstructed picture should have the same CRC value for each run because the required algorithm is deterministic.

# 4 Tests

The CTG provides two groups of testing. The first test group provides a series of pictures at specific resolutions which were chosen and/or designed to exercise a large portion of the DSC encoder and decoder logic. Additional pictures are provided to specifically stress the rate buffer implementation.

The second test group is discussed in Section 5 and provides additional coverage, if needed. This group has automated tools to allow for testing different resolutions and orientations.

## 4.1 Picture-specific Tests

The pictures and tests in this test group are designed to cover recommended PPS values, as suggested in *DSC v1.1, Annex E*. Designers who choose to implement non-standard values should determine whether coverage is sufficient as is, or if they need to expand the test cases beyond those included in this document.

The test coverage shown is derived with Variable Bit Rate (VBR) mode disabled. At this time, if an implementer chooses to enable VBR mode, they should also review the coverage in that mode and add coverage, as needed.

Additionally, the supplied test pictures are generated in 8 bits per component (bpc) RGB color format, unless noted otherwise. It is left to the tester's discretion to convert the test pictures to alternate bits per component, if needed.

A recommended PPS is provided for the pictures; however, testers are encouraged to adjust the PPS to match their expected operational modes with regard to slice size, display dimensions, etc. The PPS should match the picture size. The PPS used will impact the CRC results. Therefore, the pictures should be run through the software models to generate the correct CRCs to match the desired PPS case(s). CRCs are used at each of the following points:

1 Uncompressed source picture (Stage 1 feeding into Stage 2).

2 Compressed picture (end of Stage 2, input to Stage 4).

3 Final reconstructed picture (end of Stage 4).

For the DSC core algorithm logic, changing the picture resolution and/or orientation has minimal effect on the test coverage. However, such changes may be desired to provide additional stress testing to the DSC buffers. This may be especially useful if the implementation combines the DSC rate buffer and the transport flow buffers in some manner.

For specific resizing requirements, see Section 4.3.2 for recommended methods.

## 4.2 Running Tests

Because DSC encoders and decoders are expected to always provide deterministic output that matches the DSC Software Model output, testing can be done with a few different approaches, as described in the following subsections.

### 4.2.1 Encoder

Testing the encoder requires a method to capture its compressed output bitstream. The bitstream could then be compared to a known correct compressed output bitstream generated from the same uncompressed source picture and parameters. This good test bitstream can be generated either from the DSC Software Model or from a known good encoder.

Alternately, the compressed bitstream could be sent through a decoder (either the DSC Software Model or a known good decoder) and the reconstructed output could be compared with the known good reconstructed picture from either the DSC Software Model or a known good encoder/decoder hardware.

The DSC Software Model outputs a .dsc file with the compressed bitstream for the uncompressed source file. This same file is then used by the DSC Software Model decoder to reconstruct the output picture. For further details regarding DSC Software Model use, see Section 5.

### 4.2.2 Decoder

Testing the decoder requires a compressed bitstream that has been previously generated, either from the DSC Software Model or from a known good encoder using the same uncompressed source picture and parameters. The reconstructed output from the decoder is expected to match the reference reconstructed output from another *golden* decoder or the DSC Software Model for the decoder. Some graphics devices apply additional adjustments to the output picture between the frame buffer and the display. In those cases, the unmodified picture direct from the frame buffer might be needed in the comparison (i.e., the direct output from the decoder).

A common way to verify that the compressed bitstream (for the encoder) or the reconstructed picture (for the decoder) match the test references is to calculate a CRC value for both the test reference and the generated outputs. One CRC calculation method is included in the DSC CTG Buffer Conformance Tool. However, any CRC standard method can be used, as long as it is consistent for both the device under test and the test reference data.

It is important to understand that the DSC Software Model does not include any modeling of the actual transport. The transport is likely to map the bit ordering within the stream in a specific way, include additional control bytes, etc. Therefore, one of the following should be done:

- Capture the compressed bitstream before or after the transport is involved (that is, before the transport receives it or after the compressed bitstream is recovered from the transport as it enters the decoder), or
- Enhance the DSC Software Model to also model the transport in use

## 4.3 DSC Algorithm Core Coverage

For the DSC core algorithm logic, changing the picture resolution and/or orientation has minimal effect on test coverage. Therefore, running the pictures listed in Table 4-1, in their supplied size and orientation, should provide high coverage. See Appendix D for core test coverage information for each picture. See Section E.1 for a representation of each picture.

The NG_Shore picture provides most of the core test coverage while the CT_over_a25_N16 picture fills in specific additional coverage areas, as noted in Table 4-1. These pictures should be run with slice sizes equal to full picture width by 8 lines. It should be noted that the 8-line slice height is used only to ensure specific test coverage. The minimum line height for a slice, as provided in *DSC v1.1*, should be honored for all standard (nontest) usage.

For designs that support resolutions where having 1 slice/line will fit within their supported bandwidth maximum, run the tests with the pictures in their original dimensions. For the higher resolutions that require more than 1 slice/line, see Section 4.3.1 for the method required.

**Table 4-1: DSC Algorithm Test Pictures**

| Picture | Resolution | Coverage Notes |
|---|---|---|
| NG_Shore | 1920x1200 | Widest algorithm path coverage. |
| CT_over_a25_N16 | 592x390 | Improves flatness path coverage. |

### 4.3.1 Multiple Slices/Line Coverage

For implementations that support more than 1 slice/line, tests should include one of the prior test pictures, but with a slice distribution to match the possible configurations. For example, if a design can support 1, 2, and 5 slices/line, the chosen test picture should be run two additional times (1 slice/line was already covered in the prior section testing) with slices configured to be 2 slices/line and 5 slices/line.

### 4.3.2 Adjusting the Default Picture Size to Fit Specific Display Limitations

While the coverage report is based on the picture resolution and slice sizes provided in this document, changing the picture sizes should not have a substantial impact on test coverage. If the system under test cannot support the supplied resolutions, it is possible to adjust the picture and tile sizes to the system-required resolutions. The following guidelines are provided to help maintain test coverage as near to the uncompressed source picture as possible.

1 If the picture is larger than the display can accommodate, resize the picture to fit the display.

2 If the picture is smaller than the display can support:

   a Tile the picture into a grid of pictures, using the original-sized picture for each grid (e.g., one picture becomes a set of four pictures in a 2x2 grid).

   b Crop the new tiled picture construct to match the resolution that can be supported.

3 Adjust the slice size, as needed:

- If the uncompressed source picture slice width is one full picture width, use the new dimensions to maintain the full picture width
- If the uncompressed source picture slice width is a fraction of the picture's width, set the slice width to the same fraction of the resized picture's width

- Keep the number of lines per slice as close to the uncompressed source picture's lines per slice as possible
    - It is preferred to try to avoid decreasing the line count per slice

Appendix B provides sample command lines for using ImageMagick to achieve the above items. Other methods can also be used to achieve the same results.

## 4.4 Slice Isolation Coverage

If one or more bits becomes corrupted within a single slice in the input compressed stream for the decoder, the remainder of the same slice will be corrupted at the output picture. In proper operation, the remainder of the slices will be isolated and show no corruption. The decoder must also not hang due to such corruption.

This CTG package supplies a compressed stream that was originally generated from the NG_Shore test picture with a slightly different parameter set. The slices for this stream are configured to be 480 pixels wide by 8 pixels tall. The stream corrupts four separate bytes with single-bit errors – one each in two separate streams and two in a third stream. The resulting picture (after going through the encoder) should indicate that three of the slices are corrupted and have no impact on any other slices. Notes are included on the corrupted decoded example picture file:

- **DSC Compressed Stream File - no errors** – NG_Shore – good.dsc
- **DSC Compressed Stream File - 4 bit errors** – NG_Shore - 4 corrupted bits.dsc
- **DSC Output File resulting from 4 bit error compressed file** – NG_Shore - 4 corrupted bits - out.ppm

It is up to the Transport specifications that reference DSC to include their own tests for containment of dropped or inserted data into the compressed bitstream if such deletions or insertions are possible.

## 4.5 Buffer Control Coverage

The first picture supplied (Buffer_Stress) for buffer flow control testing is specifically designed to stress the buffer flow control by causing the DSC buffers to fill and empty in different patterns. Therefore, the picture should be run in its uncompressed format and with slices set to 1280 pixels x 480 lines.

For the other buffer test pictures, they can be run with different resolutions, picture orientations, and/or slice sizes. This may provide better buffer test coverage because buffer flow control can be impacted by changing these factors. They may be more important for implementations that combine the DSC flow buffers with the transport flow buffers and the associated control. Section 5 provides a tool to assist with variable options. See Section E.1 for a representation of each picture.

**Table 4-2: DSC Buffer Flow Control Test Pictures**

| Picture | Resolution | Slice Size | Coverage Notes |
|---|---|---|---|
| Buffer_Stress | 2560x1920 | 1280 pixels x 480 lines | Run at its original resolution and orientation. |
| StarNoise | Variable | Variable | Generated by test tool. |
| ColorBars | Variable | Variable | Generated by test tool. |

# 5 DSC CTG Buffer Conformance Tool

The DSC CTG Buffer Conformance Tool provides scripts that are used to generate certain pictures at multiple resolutions and orientations. This test group may help cover additional corner cases, especially if combined with transport buffer testing. As with the prior test section, the DSC Software Model can be used to verify a hardware design.

The files pertaining to the DSC CTG Buffer Conformance Tool are located within their own folder, supplied with this CTG. The folder contains a README.TXT file that provides complete usage instructions.

Figure 5-1 illustrates the detailed process flow driven by a CTG buffer flow spreadsheet.

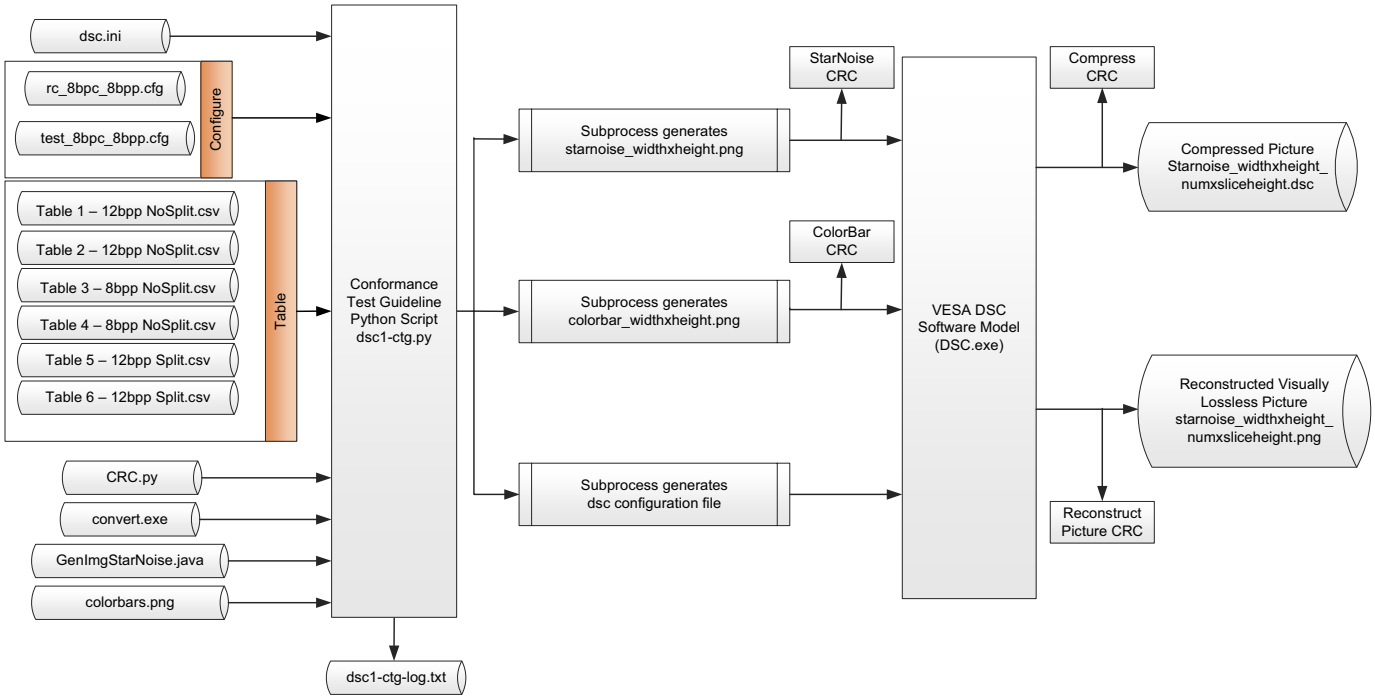


**Figure 5-1: Detailed DSC CTG Buffer Conformance Process Diagram**

The entire process is automated by a single Python script, dsc1-ctg.py. The script accepts the inputs listed in Table 5-1.

**Table 5-1: dsc1-ctg.py Script Inputs**

| Input | Description |
|---|---|
| colorbars.png | Base picture for generating colorbar picture pattern. |
| CRC.py | Python script to calculate the cyclic redundancy checks (CRCs).<br>The included file uses CCITT CRC-32, by default. An alternate CRC method may be used by updating this script to the preferred alternate. |
| dsc.ini | Contains the values for parameters such as picture type, .csv filename, compression size, input folder name, output folder name, log filename, etc. |
| GenImgStarNoise.java | Java script used to generate the StarNoise pattern. |
| **Configure** {folder} | Contains the DSC configuration files. |
| **Table** {folder} | Contains table files in .csv format saved from the CTG buffer flow spreadsheet. |

The Python script uses these inputs to drive the DSC.exe VESA DSC Software Model. The script generates an output log file for diagnostic purposes, which allows users to create and debug custom workflows. The appropriate CRCs are generated at critical points within the flow (uncompressed, compressed, and reconstructed pictures) to allow rapid comparison of results with the CTG buffer flow spreadsheet and digital simulations.

The following sections provide an overview of how this tool set is used. A sequence in which to run the Python script that controls this is included in Section 5.5.

## 5.1 How to Use the CTG Buffer Flow Spreadsheet

The input “Table {XX} – {8bpp/12bpp} {Split//NoSplit} [description].csv” configuration files in the **Table** folder drive the configuration of the DSC model. Each configuration file corresponds to a unique **Table** tab specified in the CTG buffer flow spreadsheet (described in Section 5.1). The .csv files contain the width, height, and other DSC configuration parameters that are required to generate the complete CTG data suite.

The first tab of the CTG spreadsheet, **Overview**, contains a summary of six different compression configurations that may be applicable for VESA DSC use.

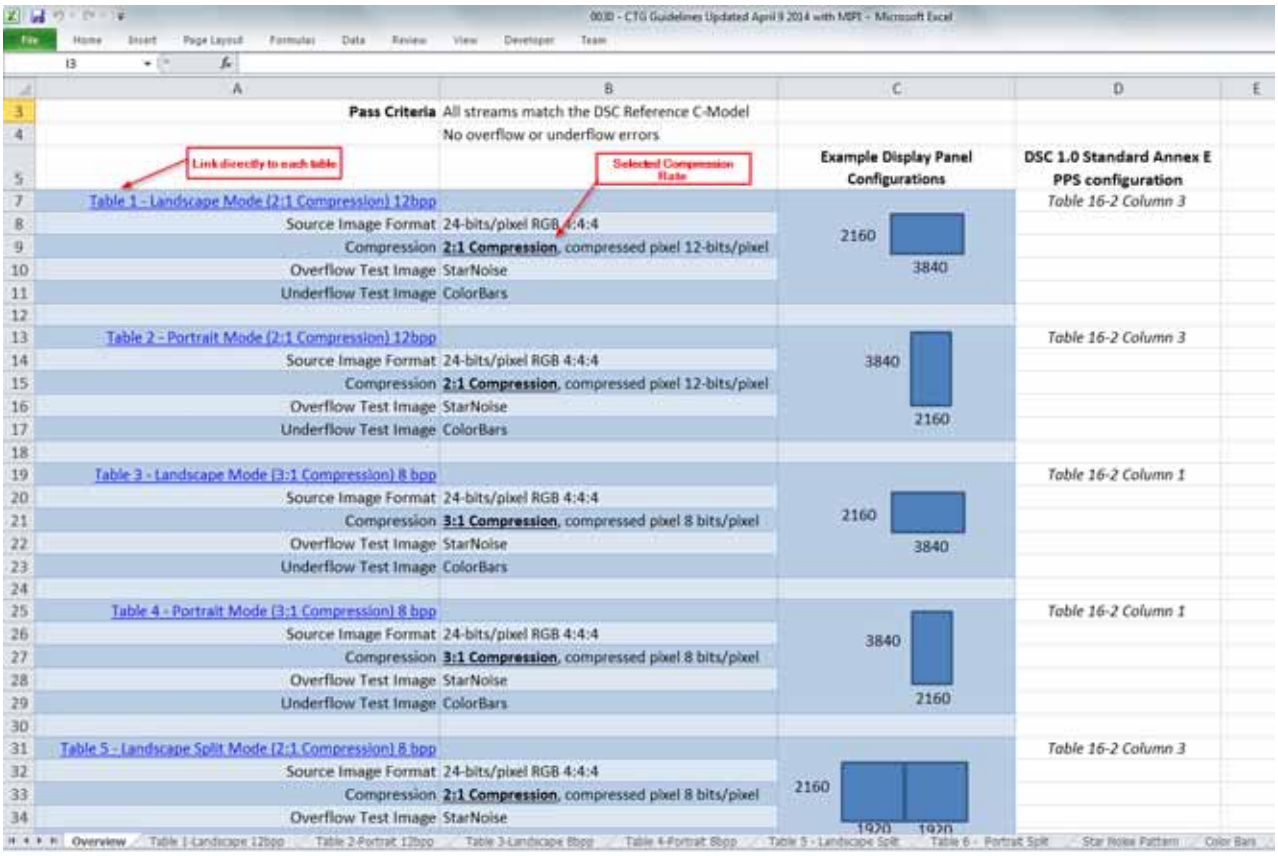


**Figure 5-2: Overview Tab**

The first worksheet tab provides an overview table with links to all six configurations. Table 5-2 lists the parameters that are evaluated in the CTG.

**Table 5-2: CTG Buffer Flow Spreadsheet Parameter Overview**

| Parameter Metric | Description | Comment |
|---|---|---|
| Pixel Format | Only a 24-bit RGB 4:4:4 pixel format is defined in the CTG. | RGB is a widely used format and is therefore covered in the first release of the DSC CTG. Implementers can expand the coverage of this tool to include other formats that *DSC v1.1* supports, as needed. |
| Compression Rate | Two compression rates are tested:<br>• **2:1 Compression** – 24bpp RGB source picture down to 12bpp<br>• **3:1 Compression** – 24bpp RGB source picture down to 8bpp | This tool supports both 2:1 and 3:1 compression ratios (the two most common ratios). The tester can add additional capabilities to the script if an implementation needs wider coverage. |
| Source Picture Files | • **StarNoise** – Complex pattern used for testing codec buffer overflow conditions<br>• **ColorBar** – Simple pattern used for testing codec buffer underflow conditions | These two artificial patterns are used to test underflow and overflow conditions. The dimensions of each source picture drive a test pattern generator to produce each input. |
| *DSC v1.1, Annex E* | The configuration for each table is mapped to the specific configuration defined in *DSC v1.1, Annex E*. | |
| *DSC v1.1, Tables 5* and *6 in the CTG Buffer Flow Spreadsheet*<br>Split mode configuration | LEFT and RIGHT panel configurations | These tests are targeted to the soft-slice configurations defined in *DSC v1.1*. |

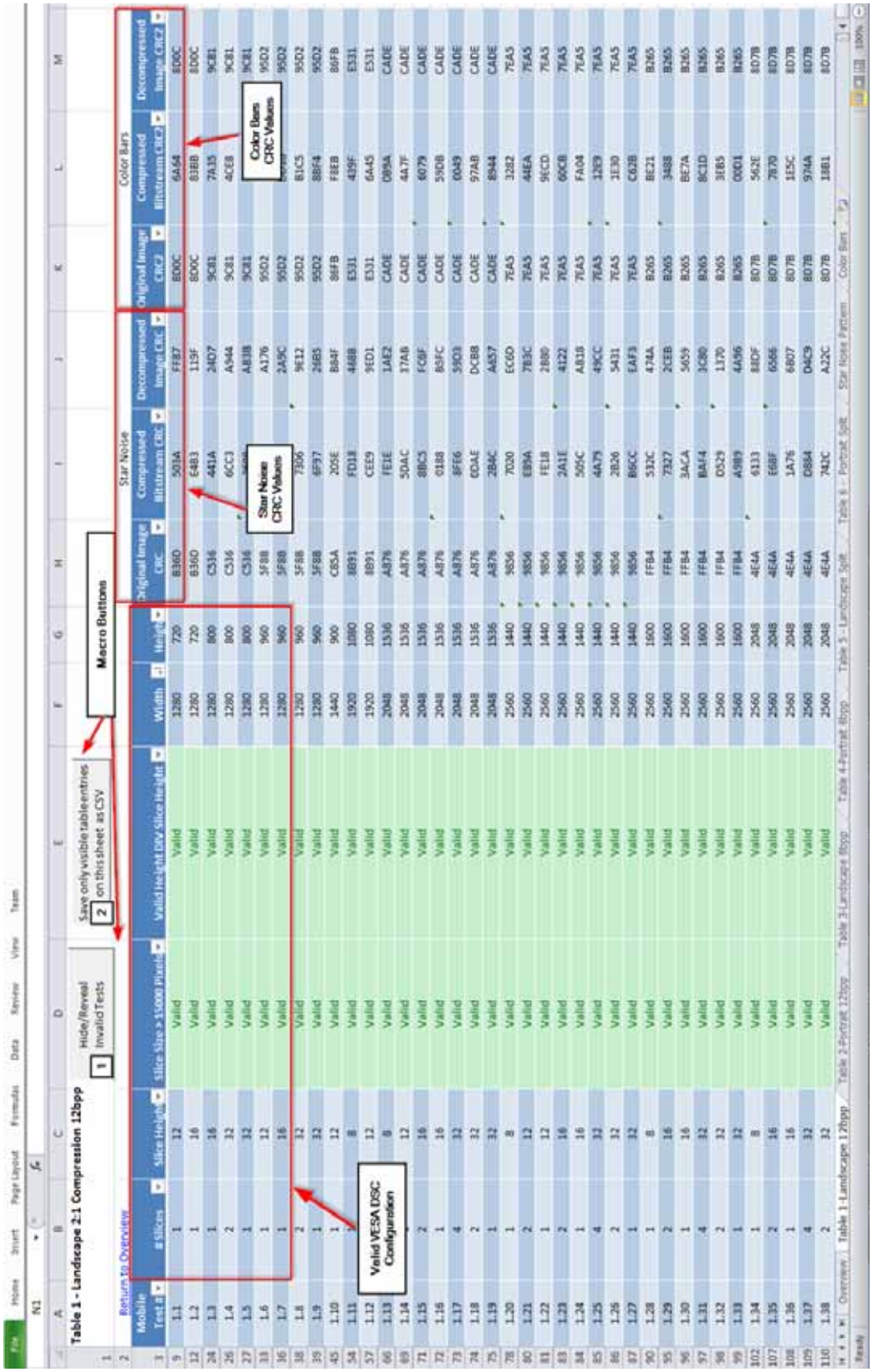


**Figure 5-3: Screen Shot from CTG Buffer Flow Spreadsheet in Table 1 Worksheet**

Figure 5-3 illustrates the information provided in each of the tabs of the VESA CTG. The test numbers correspond to specific configurations in the supplied table configuration parameter files, and this spreadsheet contains all width and height values defined in each test. Table 5-3 describes how to use the macro buttons shown in Figure 5-3.

**Table 5-3: CTG Buffer Flow Spreadsheet Macro Buttons in Figure 5-3**

| Macro Button | Label | Function |
|---|---|---|
| 1 | **Hide/Reveal Invalid Tests** | Toggles between showing all valid and invalid tests. Defined and valid VESA CTG tests require both VALID parameters in Columns D and E, as per *DSC v1.1*. |
| 2 | **Save only visible entries on this sheet as CSV** | Saves the visible rows in the table's Column B to G on the current worksheet as a .csv file, using the worksheet name as the filename, in the same folder as the CTG buffer flow spreadsheet. |

The following lists the Worksheet Naming Convention (e.g., "Table X – 12bpp Split [Description]"):

- Separate each part with a space
- The first word is always "Table"
- Followed by a number
- Then a dash: "-"
- The bpp value
  - **2:1 compression** – Use *12bpp*
  - **3:1 compression** – Use *8bpp*
- Indicate whether to use splitting
  - **Splitting** – Use *Split*
  - **No Splitting** – Use *Nosplit*
- Any description name such as "Landscape" or "Portrait" is optional onward

***Note:*** *The worksheet should not have any blank spaces at the beginning or end of its name. For example, "Table 6 - 12bpp Split Portrait " is not allowed because there is a space at the end. However, "Table 6 - 12bpp Split Portrait" and "Table 7 - 8bpp NoSplit Custom Portrait" are valid worksheet names.*

## 5.2 DSC CTG Buffer Conformance Tool Folders

Figure 5-4 illustrates the DSC CTG Buffer Conformance Tool Python script folder structure.

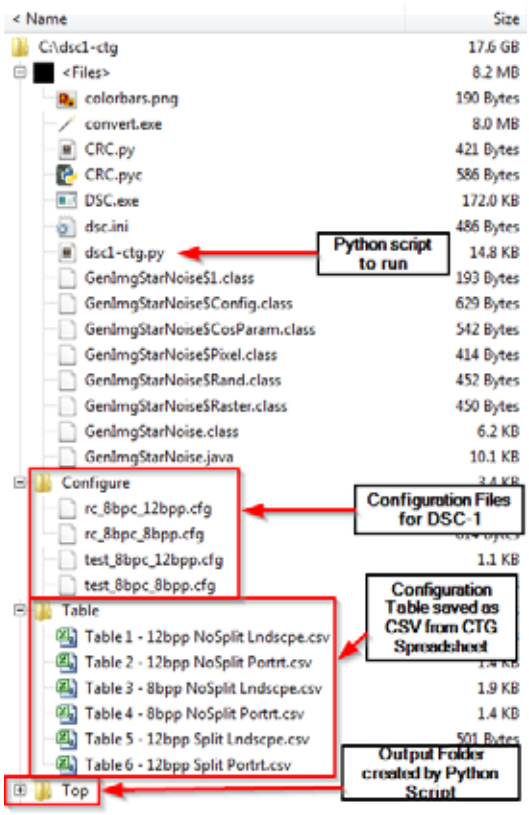


**Figure 5-4: DSC CTG Buffer Conformance Tool Python Script Folder Structure**

Figure 5-5 illustrates the **Top** output folder structure created by the Python script.

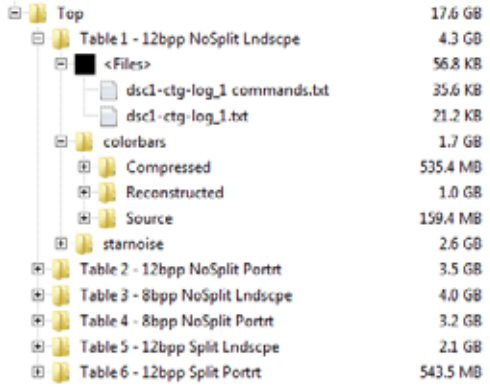


**Figure 5-5: Output Folder Structure**

There is a folder for each table. Within each **Table** folder, there are folders for each picture type and two log files – one for the CRC values and one for the commands used to call the DSC encoder and decoder.

Figure 5-6 illustrates an example of the types of files generated within the **Source** folder.

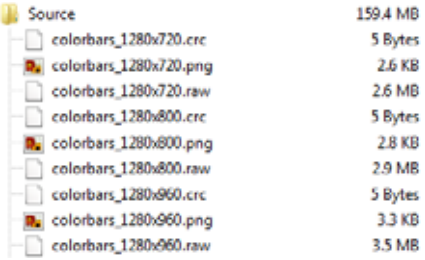


**Figure 5-6: Source Folder Structure**

For each dimension generated, three types of files are saved in the Source folder – .png, .raw, and .crc. The .crc file contains the picture's CRC value.

When the splitting is used, the left and right halves of the picture are generated, as illustrated in Figure 5-7.

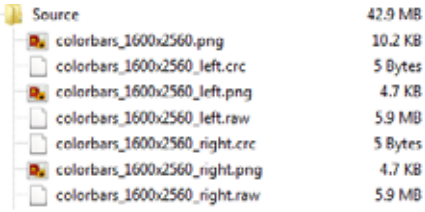


**Figure 5-7: Source Folder Structure after Splitting**

***Note:*** *With the Splitting option, the uncompressed source picture does not have a .raw or .crc file associated with it, because only the left and right halves are passed to the DSC encoder and decoder. The dimensions in the filename correspond to the uncompressed source picture's dimensions. In Figure 5-7, the "left" picture's dimensions would be 800x2560 pixels.*

Figure 5-8 illustrates an example of the types of files generated within the **Compressed** folder.

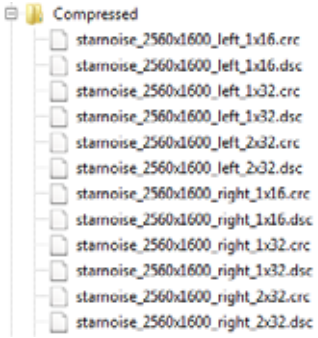


**Figure 5-8: Compressed Folder Structure**

For each compression parameter (number of slices and slice height) generated, two types of files are saved in the **Source** folder – DSC and CRC. In this case, the Splitting option was used, resulting in left and right versions.

Figure 5-9 illustrates an example of the types of files generated within the **Reconstructed** folder.

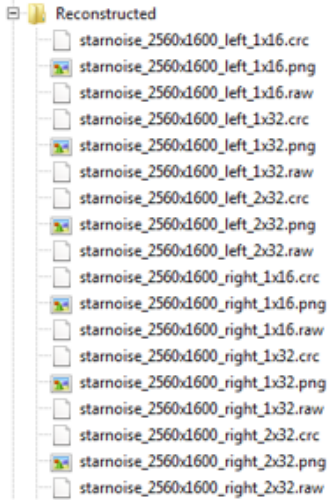


**Figure 5-9: Reconstructed Folder Structure**

For each compression parameter (number of slices and slice height) generated, three types of files are saved in the **Source** folder – .png, .raw, and .crc. Again, in this case the Splitting option was used, resulting in left and right versions.

## 5.3 DSC CTG Buffer Conformance Tool dsc.ini File

The dsc.ini file contains the necessary parameters for use with the DSC CTG Buffer Conformance Tool. Table 5-4 lists the headings and options contained within those headings.

**Table 5-4: Parameter Explanation for dsc.ini File**

| Heading | Parameter | Example Value | Description |
|---|---|---|---|
| TopDirName | `Top` | Top | Name of the folder that contains the script output. |
| SubDirName | `Src` | Source | Name of the folder that will contain uncompressed source pictures. |
| | `Comp` | Compressed | Name of the folder that will contain the compressed pictures. |
| | `Reconst` | Reconstructed | Name of the folder that will contain the reconstructed pictures. |
| InputDirName | `Config` | Configure | Name of the folder that contains the DSC configuration files. |
| | `Table` | Table | Name of the folder that contains the test configuration table .csv files. |
| TableName | `table` | Table 1 - 12bpp NoSplit Lndscpe.csv | List of the configuration table .csv filenames, separated by a comma with no space in between the commas. |
| TestPattern | `picturetype` | starnoise<br>colorbar<br>starnoise,colorbar | Type of picture test patterns. Starnoise or colorbars are the two available options. If the user needs to test both patterns, separate them by a comma with no space. |
| Bpc | `bpc` | 8bpc | Bits per component, referring to the color depth. The only available option is 8bpc. |
| LogName | `logfile` | dsc1-ctg-logfile.txt | Name of the log file that contains the CRC values for each test scenario. |

Figure 5-10 illustrates a sample dsc.ini file.

```
[TopDirName]
Top=Top

[SubDirName]
Src=Source
Comp=Compressed
Reconst=Reconstructed

[InputDirName]
Config=Configure
Table=Table

[TableName]
table=   Table 1 - 12bpp NoSplit Lndscpe.csv,
         Table 2 - 12bpp NoSplit Portrt.csv,
         Table 3 - 8bpp NoSplit Lndscpe.csv,
         Table 4 - 8bpp NoSplit Portrt.csv,
         Table 5 - 12bpp Split Lndscpe.csv,
         Table 6 - 12bpp Split Portrt.csv

[TestPattern]
picturetype=starnoise,colorbars

[Bpc]
bpc=8bpc

[LogName]
logfile=dsc1-ctg-log
```

**Figure 5-10: Example dsc.ini file**

## 5.4 StarNoise and ColorBar Configuration Files

The StarNoise and ColorBar input picture files are used to stress the rate buffers in the DSC encoder and decoder implementations. These two configuration files are created by the script to generate the input test pattern pictures corresponding to the display resolution under test. These test pictures are read by the DSC.exe VESA model as source pictures. A CRC value is generated for each input picture to ensure proper conformance validation and match with the CTG buffer flow spreadsheet.

The Python script produces these reconstructed picture files along with proper CRC values to match the CTG buffer flow spreadsheet configurations. These reconstructed pictures should match the reconstructed picture output of the digital simulations, including all provided CRC values.

## 5.5 Running the DSC CTG Buffer Conformance Tool Python Script

Before running this script, it is important to verify the correct setting for the DPX_BUGS_OVERRIDE that is used by the DSC Software Model called by the script. (See Section F.5 for further details regarding this topic, including how to confirm the setting.) If needed, the script should be modified in each location that uses the DPX_BUGS_OVERRIDE setting.

To run the DSC CTG Buffer Conformance Tool Python script, do the following:

1 Install Java, ImageMagick, and Python, if not already installed, as specified in Appendix A.

2 Locate the **Buffer_Conf_Tool** folder in the file set supplied with this CTG, and then copy the folder to your working space. This new folder will be referred to as the "root directory".

3 Create a folder named **Configure** (or as defined in your .ini file) in the root directory.

4 From the **DSC_Model** folder, copy the .cfg file(s), as needed, into the **Configure** folder. For each copied .cfg file that matches the bpc/bpp format entries, create another .cfg file that matches the one shown in Section F.9. The new filename(s) should start with "test_" instead of "rc_", but otherwise match the copied "rc_" filename(s).

5 Copy DSC.exe into the root directory.

6 Copy convert.exe from the ImageMagick install folder to the Python script folder. (Alternatively, include ImageMagick in your system's PATH environment variable.)

7 The CTG tool package includes sample .csv files inside the **Table** folders (located under the **Buffer_Conf_Tool** folder), which were generated from the CTG Guidelines spreadsheet. New .csv files can be generated for additional tests, if needed.

8 Modify the dsc.ini file, as needed.

***Note:*** *This toolset includes a sample dsc.ini file that can be modified, as needed, to include other .csv test files.*

9 Open the command line/terminal tool, then navigate to the root directory.

10 Compile the GenImgStarNoise.java file by typing `javac GenImgStarNoise.java` and then pressing Enter. A GenImgStarNoise.class file and other intermediate files should be created.

11 Run the Python script by typing `python dsc1-ctg.py`, and then pressing Enter. After a few seconds, some messages will appear, indicating that the script is making progress. (See Figure 5-11.)

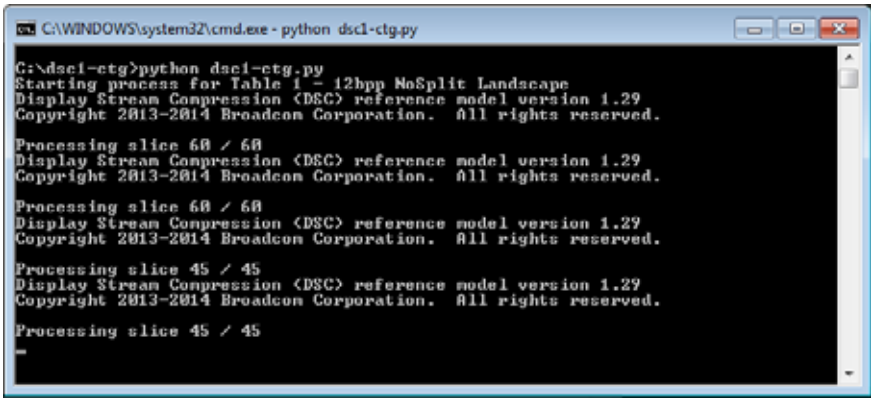


**Figure 5-11: CTG Python Script Command Line Operation**

***Note:*** *This script assumes that Python version 2.7.7 is installed. It may or may not work, as is, with other Python versions. See Appendix A for further details.*

# A Installing Software Required for Tools

The following software/utilities are used by the tools described in this CTG document. Installation and operation vary between systems; therefore, be sure to check and follow installation instructions related to the OS in use. To help minimize the test system debug, it is recommended that each tool be tested and confirmed to be operational before attempting to run any of the scripts identified in this CTG package.

## A.1 Python

The Python scripts have all been tested against Python version 2.7.7. The scripts will likely not work with Python version 3.*n* due to substantial changes in the newer Python builds. Python downloads are available from www.python.org.

The Python executable should be included in the system's PATH environment variable.

## A.2 Java JDK

The GenImgStarNoise.java file is provided. Java is preinstalled on many systems.

For systems that need to have Java installed, it can be downloaded from www.java.com.

The Java executable should be included in the system's PATH environment variable.

## A.3 Installing ImageMagick

ImageMagick can be downloaded from www.imagemagick.org.

The primary function used for these tools is convert.exe. You can either ensure that ImageMagick is in the system's PATH environment variable or alternately, copy "convert.exe" to the same folder as the dsc1-ctg.py Python script.

***Note:*** *Windows versions appear to accept the* `convert.exe` *or* `convert` *command. MAC OS X systems will accept only the* `convert` *command.*

# B Usage Notes on ImageMagick for Picture Sizing

The following command line examples show how ImageMagick can be used to modify a picture if needed. Other programs can be used to accomplish the same effect. The `NG_Shore.ppm` picture is used for the reference. By default, the picture file is 1920x1200 pixels in size.

To reduce the picture size to 1200x960 pixels, the **convert** command can be used, as shown below. Because the new dimensions do not preserve the uncompressed source picture's aspect ratio, a `!` character should be used after the dimensions in the command line (e.g., `1200x860!`) to provide the final size.

```
convert NG_Shore.ppm -resize 1200x960! NG_Shore_1200.ppm
```

To make a 2x2 tiled copy of the uncompressed source picture (two pictures wide and two pictures tall = 3840x2400) and then crop the picture to 2560x1600, use the **montage** and **convert** commands, as follows:

```
montage NG_Shore.ppm NG_Shore.ppm NG_Shore.ppm NG_Shore.ppm -
  geometry 1920x1200 -tile 2x2 montage.ppm
convert montage.ppm -crop 2560x1600 FinalFile.ppm
```

The **montage** command's `–geometry` parameter is needed to maintain the original size of each picture resulting in the montage.ppm output picture. The command's `–tile` parameter provides the matrix geometry (which could also have been 1x4 or 4x1 for the four input pictures).

The **convert** command crops the input montage.ppm picture, providing the final result in the FinalFile.ppm picture.

For further details regarding ImageMagick command usage, see the ImageMagick documentation.

# C CTG File Summary

Table C-1 lists the DSC Software Model and related files that are included in the software package provided with *DSC v1.1*. The software package is included with *DSC v1.1*.

**Table C-1: DSC C Model Files**

| Filename | Description |
|---|---|
| DSC.exe | Main executable file. |
| rc_xxbpc_yybpp.cfg | DSC baseline configuration files. xx = 8, 10, 12; yy = 6, 8, 10, 12, 15. |
| README.TXT | Instructions for usage. |
| Source {folder} | Folder that contains the DSC C model source files. |
| test_xxbpc_yybpp.cfg | Test sample baseline configuration files. |
| test_list.txt | List of pictures to be processed by the DSC code. |
| test.cfg | Main configuration file used by the DSC code. Calls out the DSC baseline configuration files. |

The files listed in Table C-2 through Table C-4 are included in the DSC CTG files package that is included with this document.

Table C-2 lists the picture files that are used in the testing outlined in Section 4.2 and Section 4.4. These files are located in the **Pictures** folder.

**Table C-2: DSC CTG Algorithm Core Coverage Files**

| Filename | Description |
|---|---|
| Buffer_Stress | Picture to exercise the buffers that test the buffer control. |
| CT_over_a25_N16 | Picture that adds additional flatness path coverage. |
| NG_Shore | Picture that provides the widest algorithm coverage. |

Table C-3 lists the compressed stream and corrupted output files used in DSC Slice Isolation Coverage. (See Section 4.3.) These files are located in the **Slice_Isolation** folder.

**Table C-3: DSC CTG Slice Isolation Files**

| Filename | Description |
|---|---|
| NG_Shore - good.dsc | Reference good compressed stream file capture from NG_Shore (slices = 480 pixels x 8 lines). |
| NG_Shore - 4 corrupted bits.dsc | Corrupted compressed stream file capture from NG_Shore (slices = 480 pixels x 8 lines). |
| NG_Shore - 4 corrupted bits - out.ppm | Sample decoded picture from corrupted NG_Shore compressed stream file with notes regarding corruption areas. |

Table C-4 lists files and a folder included in the software package that are used only when running the DSC CTG Buffer Conformance Tool. (See Section 5.) These files and folder are located in the **Buffer_Conf_Tool** folder.

**Table C-4: DSC CTG Buffer Conformance Tool Files**

| Filename | Description |
|---|---|
| colorbars.png | Uncompressed source picture for ColorBars testing. Used by the script to generate different size/proportion variants. |
| CRC.py | Python script that generates the CRC. Used by the dsc1-ctg.py script. An alternate CRC method can be used by modifying this script. |
| CTG Guidelines 16062014.xlsm | Spreadsheet that controls the tests/formats with which dsc1-ctg.py tests. |
| dsc1-ctg.py | Main python script that is used to run this tool. |
| dsc.ini | Control file used for the DSC CTG Toolset script. |
| GenImgStarNoise.java | Java code that generates the StarNoise test pictures. Used by dsc1-ctg.py. |
| license.txt | License under which the tool may be used. |
| **Table** {folder} | Folder with six different tables that are used with the DSC CTG Toolset script. |

# D DSC Encoder/Decoder Picture Coverage Information

## D.1 Core Algorithm Coverage

Table D-1 provides the coverage of different paths through the core algorithm of the pictures shown when run with the slice sizes as notes. The **Maximum Paths** column indicates how many different routes can happen through the Parameter's code path.

It is important to note that due to the algorithm's design, some of the possible paths are not expected to actually run. For example, in the case of VBR_underflow, this will happen only if there is a buffer underflow; however, this should be impossible with a proper design when running with VBR disabled. Because the test coverage was generated based on the supplied DSC Software Model with VBR disabled, that loop was never taken.

In Table D-1, **green** table cells with a ✔ indicate full (100%) coverage of all possible runs through the parameter's code section. All other cases show the specific parameter routes that are not covered. Where there is a dash between numbers this is inclusive (1 – 4 is the same as 1 2 3 4). For entries that are done as "x 2 lines", these are fairly high count items (which relates to the higher possible number of Maximum Paths for that parameter). A "x 1 line" entry has better coverage than a "x 3 lines" entry.

**Table D-1: Picture Coverage for Core Algorithm**

| Parameter | Maximum Paths | NG_Shore | CT_over_a25_N16 |
|---|---|---|---|
| Quant_table_Luma | 16 | ✔ | 14, 15 |
| Quant_Table_Chroma | 16 | ✔ | 14, 15 |
| Residual_Sizes | 9 | ✔ | 0, 1 |
| Pred_Size_Adj | 100 | x 1 line | x 2 lines |
| NZ_Midpoint | 8 | ✔ | 6, 7 |
| BP_Search[a] | 8 | ✔ | ✔ |
| Enc_Flatness_Chk | 5 | 2 | ✔ |
| N_Adjustment_Bits | 9 | 1 – 8 | 1 – 8 |
| Scale | 56 | x 1 line | x 1 line |
| RC_Ranges | 15 | 13, 14 | 11 – 14 |
| RC_Paths | 10 | ✔ | 7, 9 |
| VLC_Paths | 9 | ✔ | ✔ |
| Force_MPP | 2 | ✔ | ✔ |
| MPP_Bounding | 2 | ✔ | 1 |
| ICH_Use | 192 | ✔ | ✔ |
| Flat_QP_Adj[a] | 2 | ✔ | ✔ |
| VBR_Underflow | 1 | 0 | 0 |
| BP_Modes | 8 | ✔ | 0 – 7 |

*a. When Block Prediction (BP) is disabled in the .cfg files, `BP_Search` becomes `0 – 7` and `Flat_QP_Adj` becomes 1.*

## D.2 Buffer Stress Coverage

Table D-2 lists the two recommended pictures to use for buffer stress testing and the minimum and maximum buffer model levels that they hit per slice. The rbsMin value is the maximum that could, in theory, be hit; however, there is a control in the code that prevents the buffer from completely filling up to this point. Additionally, a buffer can never be allowed to go below 0.

The minimum buffer model fullness per slice is based on the minimum fullness after the ***initial_xmit_delay***. It does not include the initial buffer model fullness, which is always 0 per slice (assuming that the DSC and the transport buffers are implemented independently; some implementations may combine the two and consideration should be made for the impact of such a design in the testing).

Both pictures provide fairly good high-end coverage; however, the Buffer_stress picture is the only one that fully empties the buffer on some of the slices. Changing the picture's slice height to 192 lines will provide a higher number of fill/empty cases due to the picture design. However, changing the height will not test for additional corner cases, and actually has a slightly lower maximum buffer fill value for the full picture. Therefore, such a change is not recommended for this test function.

**Table D-2: Buffer Model Full/Empty Coverage for Buffer Stress and StarNoise Testing**

| | Buffer_stress_2560x1920x9bpc Slice = 1280 pixels x 480 lines | | Starnoise_00000_640x480 Slice = 320 pixels x 48 lines | |
|---|---|---|---|---|
| | **Minimum** | **Maximum** | **Minimum** | **Maximum** |
| **rbsMin** | | 16641 | | 11841 |
| **% full** | | 93.6% | | 85.6% |
| **For Picture** | 0 | 15583 | 1417 | 10130 |
| **Slice #** | | | | |
| 1 | 13 | 12999 | 3205 | 10107 |
| 2 | 3 | 15583 | 2517 | 10130 |
| 3 | 13 | 14720 | 2973 | 9899 |
| 4 | 2 | 15579 | 1557 | 7228 |
| 5 | 1 | 13002 | 2860 | 9475 |
| 6 | 1 | 15582 | 1680 | 6708 |
| 7 | 0 | 12124 | 2564 | 9130 |
| 8 | 0 | 12224 | 2674 | 6844 |
| 9 | | | 2967 | 7837 |
| 10 | | | 1879 | 7894 |
| 11 | | | 2996 | 9271 |
| 12 | | | 1417 | 6880 |
| 13 | | | 3019 | 9326 |
| 14 | | | 1443 | 6557 |
| 15 | | | 3267 | 9608 |
| 16 | | | 2587 | 6715 |
| 17 | | | 3339 | 10070 |
| 18 | | | 3342 | 8764 |
| 19 | | | 3332 | 10110 |
| 20 | | | 3339 | 10109 |

# E Test Pictures

## E.1 DSC Algorithm Core Coverage Test Pictures

Table E-1 lists the test pictures and their associated graphics for the pictures tested in Section 4.3.

**Table E-1: DSC Algorithm Test Pictures**

| Picture | Resolution | Coverage Notes | Graphic |
|---|---|---|---|
| NG_Shore | 1920x1200 | Widest algorithm path coverage. | Figure E-1 |
| CT_over_a25_N16 | 592x390 | Improves Flatness path coverage. | Figure E-2 |

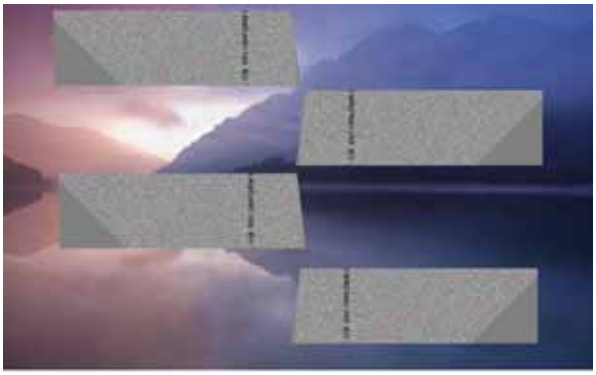

**Figure E-1: NG_Shore Test Picture Sample**


**Figure E-2: CT_over_a25_N16 Test Picture Sample**

## E.2 Buffer Control Coverage Test Pictures

Table E-2 lists the test pictures and their associated graphics for the pictures tested in Section 4.5.

**Table E-2: DSC Buffer Flow Control Test Pictures**

| Picture | Resolution | Slice Size | Coverage Notes | Graphic |
|---|---|---|---|---|
| Buffer_Stress | 2560x1920 | 1280 pixels x 480 lines | Run at its original resolution and orientation. | Figure E-3 |
| StarNoise | Variable | Variable | Generated by test tool. | Figure E-4 |
| ColorBars | Variable | Variable | Generated by test tool. | Figure E-5 |

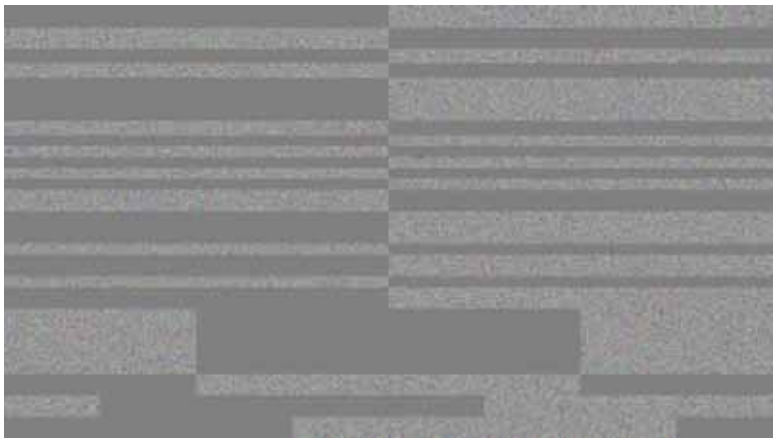

**Figure E-3: Buffer_Stress Test Picture Sample with Slice Overlay in Red**

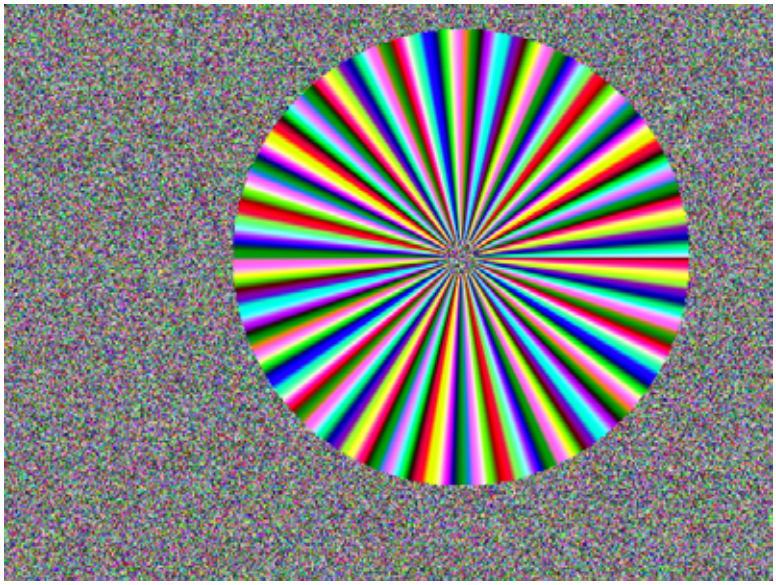

**Figure E-4: StarNoise Test Picture Sample**

**Figure E-5: ColorBars Test Picture Sample**

# F Use of .cfg Files with Test Pictures

The following file is the common .cfg file that can be used for running the different test pictures through the DSC Software Model. When testing the hardware. the .ppm file settings should reflect the same values. See the notes below the sample .cfg file source for information regarding how to set different parameters for each of the tests provided within this document.

For the Buffer Conformance Tool that generates the different formats of the StarNoise and ColorBar pictures, see that tool's documentation for correct values for the .cfg file(s).

```
// This is a config file for the Display Stream Compression model
SRC_LIST   test_list.txt  // this file holds the name of the picture
file(s) to be processed


FUNCTION  0    // 0=encode/decode (no bitstream out), 1=encode only,
2=decode only


// SLICE_WIDTH   1280  // if not set to a value, set it to width
of picture
SLICE_HEIGHT  8  // To set to one slice per picture, comment out this line


LINE_BUFFER_BPC    9
ENABLE_422     0
USE_YUV_INPUT  0        // !! Must be = 1 if 4:2:2 enabled
BLOCK_PRED_ENABLE    1  // To disable block prediction, clear this to 0
VBR_ENABLE     0       // Set to 1 for VBR mode


DPX_BUGS_OVERRIDE  0   // only needed if using .dpx files; not needed for
.ppm files
    //  Legal values for DPX_BUGS_OVERRIDE:
    //    -1 => Try to autodetect bugs (only works for some files); good
for Mac and ImageMagick
    //    0 => Standards-compliant
   //    1 => Reads 8-bit files that were generated with padded line ends
(see DPX_PAD_LINE_ENDS)
    //    2 => Fixes common issue where BGR is stored as RGB and
endianness is backwards
    //          (for example, ImageMagick output)
    //    3 => For 16-bit files, swaps the 16-bit words within a 32-bit
word
    //         DPX_PAD_LINE_ENDS  1
```

```
    // ie., start lines on 32-bit boundaries. Required to output RGB to
XNView 1.99 (but not YUV!)


    //  Select the RC settings file based on the BPC & BPP above:
    INCLUDE    rc_8bpc_12bpp.cfg
```

## F.1 SRC_LIST

SRC_LIST points to a file that holds the source picture filename. Be sure to always include the filename extension (i.e., .dpx or .ppm).

## F.2 SLICE_WIDTH and SLICE_HEIGHT

SLICE_WIDTH and SLICE_HEIGHT indicate the slice's pixel width and height, respectively. If width or height are not declared by using // in front of the starting key word, the code will automatically make the width/height to match the input picture. The sample code shows the SLICE_WIDTH not set so it equals the picture's actual width. The SLICE_HEIGHT is shown as eight lines.

***Note:*** ***SLICE_WIDTH*** *– If more than 1 slice/line is supported by a design, see Section 4.3.1 for suggested test run variants.*

**Table F-1: Slice Dimensions to Be Entered into the .cfg Files for Testing**

| Picture | Test | SLICE_WIDTH | SLICE_HEIGHT |
|---|---|---|---|
| NG_Shore | Core | 1920 *1 | 8 |
| NG_Shore | Slice Isolation | 480 | 8 |
| CT_over_a25_N16 | Core | 592 *1 | 8 |
| Buffer_Stress | Buffer Stress | 1280 | 480 |

***Note:*** *For designs that support more than 1 slice/line, see Section 4.3 for recommendations for additional SLICE_WIDTH runs to test.*

## F.3 BLOCK_PRED_ENABLE

If Block prediction is supported, run the test with BLOCK_PRED_ENABLE set to 1 and then cleared to 0. If Block prediction is not supported, only run the test with BLOCK_PRED_ENABLE cleared to 0.

## F.4 VBR_ENABLE

This document does not currently test for VBR mode; therefore, clear VBR_ENABLE to 0. If VBR mode is supported, set VBR_ENABLE to 1.

***Note:*** *If this case is enabled, it is the tester's responsibility to determine coverage.*

## F.5 DPX_BUGS_OVERRIDE

In some cases, when using a picture file in .dpx format, there can be a variance in how the .dpx Standard is interpreted. In some cases, the `DPX_BUGS_OVERRIDE` variable can be used to compensate for such variances, if needed. `DPX_BUGS_OVERRIDE` is ignored if a .ppm source file is used.

It is important to test the setting used with the versions of the file viewer and ImageMagick (or similar tool) to be used and on the specific system (Windows version, OS X, Unix, etc.) on which the software model and scripts are run. The DSC software encoder generates an interim version of the .dpx input file. By running the DPX picture with the proper override setting through the software model, the uncompressed source picture, the interim file (picture) from the DSC Software Model, and the final reconstructed picture will all appear normal without any inverted colors or other general issues.

## F.6 INCLUDE rc_…

Choose the rc_….cfg file according to input color [number of bits per component (bpc)] and total average bits in a compressed pixel (bpp). The example shown (rc_8bpc_12bpp.cfg) has a 24-bit pixel input (8bpc), which is compressed down to an average of 12bpp for transport.

## F.7 Code Sample 1 – .cfg File Used for DSC Core Test Pictures

```
// This is a config file for the Display Stream Compression model

SRC_LIST   test_list.txt

FUNCTION  0   // 0=encode/decode (no bitstream out), 1=encode only,
  2=decode only

// SLICE_WIDTH   1280; if not set to a value, set it to width of picture
SLICE_HEIGHT  8  // To set to one slice per picture, comment out this line

LINE_BUFFER_BPC    9
ENABLE_422     0
USE_YUV_INPUT  0     // !! Must be = 1 if 4:2:2 enabled
BLOCK_PRED_ENABLE    1 // To disable block prediction, clear this to 0
VBR_ENABLE     0     // Set to 1 for VBR mode

DPX_BUGS_OVERRIDE  1   // this is the corrected version for the Mac
  and ImageMagick
//  Legal values for DPX_BUGS_OVERRIDE:
//    -1 => Try to autodetect bugs (only works for some files)
//    0 => Standards-compliant
//    1 => Reads 8-bit files that were generated with padded line ends
  (see DPX_PAD_LINE_ENDS)
//    2 => Fixes common issue where BGR is stored as RGB and endianness
  is backwards
//          (for example, ImageMagick output)
//    3 => For 16-bit files, swaps the 16-bit words within a 32-bit word
//         DPX_PAD_LINE_ENDS  1
// ie., start lines on 32-bit boundaries. Required to output RGB
  to XNView 1.99 (but not YUV!)

//  Select the RC settings file based on the BPC & BPP above:
INCLUDE    rc_8bpc_12bpp.cfg
```

## F.8 Code Sample 2 – .cfg File Used for DSC Buffer_stress_2560x1920x9bpc

```
// This is a config file for the Display Stream Compression model

SRC_LIST   test_list.txt

FUNCTION  0   // 0=encode/decode (no bitstream out), 1=encode only,
2=decode only

SLICE_WIDTH   1280  // if not set to a value, set it to width of picture
SLICE_HEIGHT  480  // To set to one slice per picture, comment out
this line

LINE_BUFFER_BPC    9
ENABLE_422     0
USE_YUV_INPUT  0     // !! Must be = 1 if 4:2:2 enabled
BLOCK_PRED_ENABLE    1 // To disable block prediction, clear this to 0
VBR_ENABLE     0     // Set to 1 for VBR mode

DPX_BUGS_OVERRIDE  1   // this is the corrected version for the Mac
and ImageMagick
//  Legal values for DPX_BUGS_OVERRIDE:
//    -1 => Try to autodetect bugs (only works for some files)
//    0 => Standards-compliant
//    1 => Reads 8-bit files that were generated with padded line ends
(see DPX_PAD_LINE_ENDS)
//    2 => Fixes common issue where BGR is stored as RGB and endianness
is backwards
//          (for example, ImageMagick output)
//    3 => For 16-bit files, swaps the 16-bit words within a 32-bit word
//         DPX_PAD_LINE_ENDS  1
// ie., start lines on 32-bit boundaries. Required to output RGB to
XNView 1.99 (but not YUV!)

//  Select the RC settings file based on the BPC & BPP above:
INCLUDE    rc_8bpc_12bpp.cfg
```

## F.9 Code Sample 3 – .cfg File Used for DSC Starnoise_00000_640x480

```
// This is a config file for the Display Stream Compression model

SRC_LIST   test_list.txt

FUNCTION  0   // 0=encode/decode (no bitstream out), 1=encode only,
2=decode only

SLICE_WIDTH   320  // if not set to a value, set it to width of picture
SLICE_HEIGHT  48  // To set to one slice per picture, comment out
this line

LINE_BUFFER_BPC    9
ENABLE_422     0
USE_YUV_INPUT  0     // !! Must be = 1 if 4:2:2 enabled
BLOCK_PRED_ENABLE    1 // To disable block prediction, clear this to 0
VBR_ENABLE     0     // Set to 1 for VBR mode

DPX_BUGS_OVERRIDE  1   // this is the corrected version for the Mac
and ImageMagick
//  Legal values for DPX_BUGS_OVERRIDE:
//    -1 => Try to autodetect bugs (only works for some files)
//    0 => Standards-compliant
//    1 => Reads 8-bit files that were generated with padded line ends
(see DPX_PAD_LINE_ENDS)
//    2 => Fixes common issue where BGR is stored as RGB and endianness
is backwards
//          (for example, ImageMagick output)
//    3 => For 16-bit files, swaps the 16-bit words within a 32-bit word
//         DPX_PAD_LINE_ENDS  1
// ie., start lines on 32-bit boundaries. Required to output RGB to
XNView 1.99 (but not YUV!)

//  Select the RC settings file based on the BPC & BPP above:
INCLUDE    rc_8bpc_12bpp.cfg
```