DxO ISP2013
FVTS User Guide

Table of contents

Table of figures

Revision History

Revision	Date	Comments
V1.0	Oct 13, 2013	First version for DxO ISP2013

Table 1: Document Revision History

1 GLOSSARY

Acronym	Meaning
DUT	Device Under Test
FVTS	Functional Verification Test Suite
GBD	Gnu DeBugger
GNU	Gnu is Not Unix
HAL	Hardware Abstraction Layer
IPC	Imaging Processor Core
ISP	Image Signal Processor
IT	InTerruption
JTAG	Join Test Action Group
NDA	Non Disclosure Agreement
TNR	Temporal Noise Filtering

2 INTRODUCTION

2.1 Purpose

The aim of this document is to describe the verification vectors and environment delivered by DxO Labs for the purpose of pre- and post-silicon verification of DxO ISP once integrated in a chip. This environment is called FVTS standing for Functional Verification Test Suite.

2.2 Audience

This document is oriented to the engineering team who needs to verify DxO ISP integration at chip level.

2.3 Use of the present document

The present document is confidential and subject to the terms of the NDA signed between DxO Labs and the recipient of the present document.

The present document is the property of DxO Labs and must be considered as DxO Labs background intellectual property. This document discloses a number of technical characteristics protected by patents filed by DxO Labs, and therefore subject to royalties whatever the product they are used in. The content of this document cannot be used for any purpose without a contract from DxO Labs.

References to third party tools or hardware in the present document are not an authorization or recommendation by DxO Labs to use such third party tool or hardware. The use of any third party tool or hardware is the responsibility of the recipient of the present document.

The present document is preliminary and may contain errors in the description that do not reflect the actual behavior of the product. DxO Labs will update the present document if and when such errors are detected.

2.4 References

Ref[1] DxO_ISP2013-fwIntegrationGuide

3 FVTS DESCRIPTION

3.1 Presentation

The FVTS is a complete workflow delivered by DxO Labs to its customers allowing pre- and post-silicon verification. It includes pre-defined test vectors intended to be replayed at the chip vendor's side and checked against the references provided.

3.2 Overview

The FVTS is based on a global architecture that aims to provide all needs at concerned stages (validation, integration and troubleshooting). Through such an architecture, the goal is to be able to generate from a unique stimulus either test vectors, debug traces, validation vector or a mean to perform a replay between a reference software simulation and a RTL or chip level replay.

3.3 FVTS contents

A FVTS delivered to a chip vendor contains the following items:
- A set of test vectors intended to be replayed at chip vendor's side for validation purposes
- An illustration of the compilation and link of a test vector
- This current document as FVTS user guide

All these items are delivered through a tarred and zipped file called DxOISP-FW-FVTS-delivery.tar.gz.

4 FVTS TEST VECTOR

4.1 Introduction

The test vector definition assumes that the chip vendor is able to execute a standalone executable uploaded (typically through JTAG) on the system processor and to synchronize properly the sending and reception of pixels buffer to and from DxO ISP.
Please note that the tests are built from a functional perspective. Intermediate images stored in the frame buffer, AHBL communication between libDxOISP and DxO ISP are not logged. Only the external inputs and outputs are verified:
1. All pixstream inputs
2. All pixstream outputs
3. Communication to and from the sensor HAL
4. Communication to and from the camera application

4.2 Test Vector Description

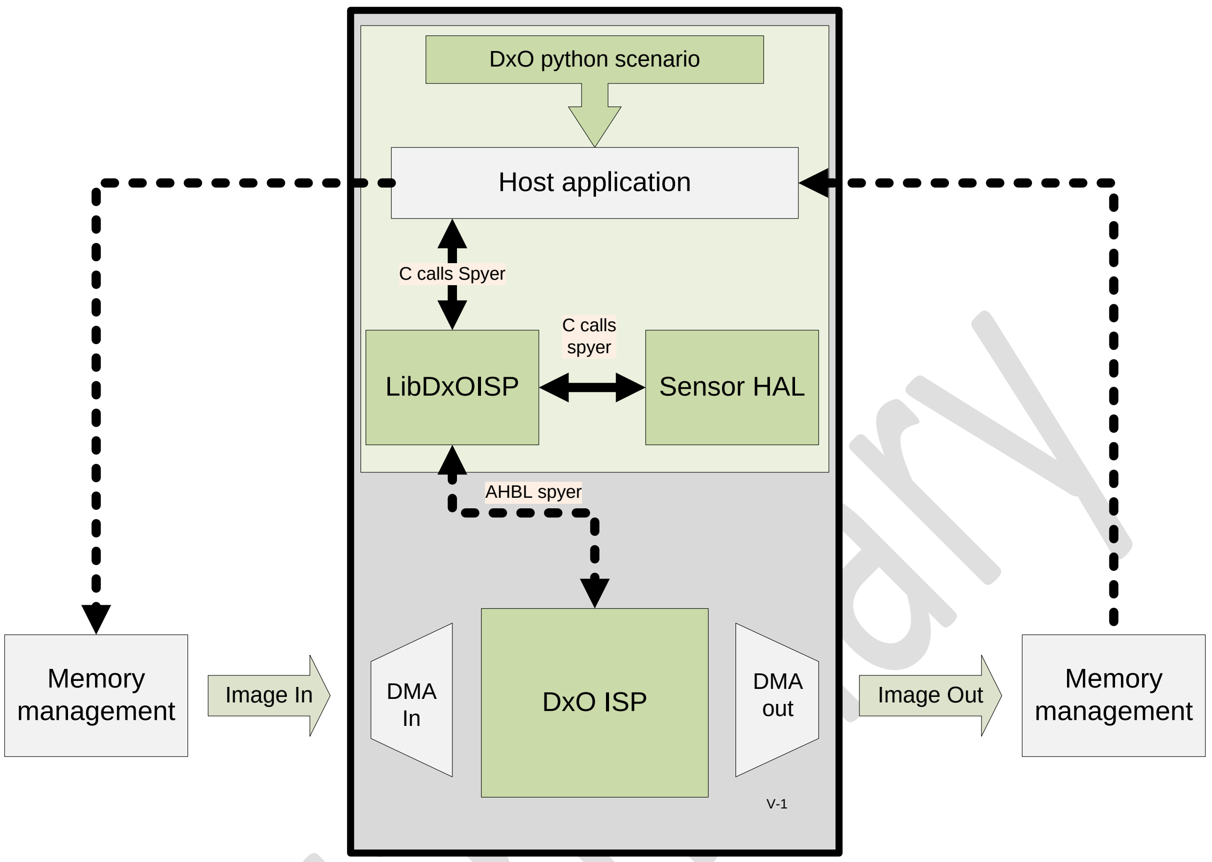

Figure 1: FVTS test bench environment

A functional scenario, initially played at DxO's site with a python environment, produces some specific traces at different levels. Each input and output are stored in a specific format allowing to generate some C programming language stimulus.

The first generated C file, named <testName>/fvtsIsp.c, is intended to simulate the communication between the camera application and the LibDxOISP (i.e. to behave like a simple camera application from DxOISP point of view).
The second one, named <testName>/sensorFvts.c, is intended to simulate a predictable behavior of the sensor HAL and does not require an actual sensor to be plugged in. Having an actual sensor makes it impossible to have repeatable runs.

FVTS test vector contents are described in the figure below. After extraction of DxOISP-FW-FVTS-delivery.tar.gz, the following content is located under: `./fvts/`

```
./readme.txt
./makefile
./src/fvtsBench.c
./src/fvtsPlayback.c
./include/fvtsPlayback.h
./<test_name>/fvtsIsp.c
./<test_name>/fvtsSensor.c
./<test_name>/input/
./<test_name>/input/<image_name>.raw
./<test_name>/output/
./<test_name>/output/<image_name>.raw
```

Figure 2: canvas FVTS test vector directory layout

fvtsPlayback.c :	This file contains the required functions able to play the vectors defined within `fvtsIsp.c` and `fvtsSensor.c`.
fvtsPlayback.h :	Header file containing prototypes of the functions that the customer must define based on its bench capacity. All functions to be defined by customer are prefixed by "`fvtsBench_`". In addition, it exports a specific function called `fvtsEventPlayback()` that has to be called by the main of the customer.
fvtsBench.c :	An example of FVTS test launcher. This is the one used at DxO's site to replay the vectors delivered to customer.
readme.txt :	brief content description of provided tests.
makefile :	fvts test compilation canvas

4.3 `fvtsIsp.c` file example

This file contains a part of the test sequence to be executed on chip vendor's platform. This file specifically contains the flow exchange to be monitored between camera application and libDxOISP. A short example of a test scenario is given below.

```
// =================================================================
// DxO Labs proprietary and confidential information
// Copyright (C) DxO Labs 1999-2013 - (All rights reserved)
// =================================================================
#include <stdint.h>
#include <stdio.h>
#include "qa/ispHw/fvts/fvtsPlayback.h"
static const uint8_t ispEventBuf_2[1] = { 0x00 };
static const CameraAppCmdSts ispEventParams_2 = {
        ispEventBuf_2                  // buf
,       0                              // offset
,       1                              // size
};
static const uint8_t ispEventBuf_5[1] = { 0x01 };
static const CameraAppCmdSts ispEventParams_5 = {
        ispEventBuf_5                  // buf
,       0                              // offset
,       1                              // size
};
static const uint8_t ispEventBuf_7[1] = { 0x00 };
static const CameraAppCmdSts ispEventParams_7 = {
        ispEventBuf_7                  // buf
,       0                              // offset
,       1                              // size
};
static const uint8_t ispEventBuf_8[4] = { 0x00, 0x00, 0x00, 0x00 };
static const CameraAppCmdSts ispEventParams_8 = {
        ispEventBuf_8                  // buf
,       8                              // offset
,       4                              // size
};
const uint32_t NbIspEvent = 8;
const CameraAppEvent IspEvents[8] = {
        { EVT_ISP_INITIALIZE            , (const CameraAppEvtData *)NULL                }
,       { EVT_ISP_STATUS_GROUP_OPEN     , (const CameraAppEvtData *)NULL                }
,       { EVT_ISP_STATUS_GET            , (const CameraAppEvtData *)&ispEventParams_2   }
,       { EVT_ISP_STATUS_GROUP_CLOSE    , (const CameraAppEvtData *)NULL                }
,       { EVT_ISP_COMMAND_GROUP_OPEN    , (const CameraAppEvtData *)NULL                }
,       { EVT_ISP_COMMAND_SET           , (const CameraAppEvtData *)&ispEventParams_5   }
,       { EVT_ISP_STATUS_GROUP_OPEN     , (const CameraAppEvtData *)NULL                }
,       { EVT_ISP_STATUS_GET            , (const CameraAppEvtData *)&ispEventParams_7   }
};

// =================================================================
// END OF FILE
// =================================================================
```

Figure 3: fvtsIsp.c file example

4.4 `fvtsSensor.c` file example

This file contains a part of the test sequence to be executed on chip vendor's platform. This file specifically contains the flow exchange to be monitored between the sensor HAL and libDxOISP. A short example of a test scenario is given below.

```
// ======================================================================
// DxO Labs proprietary and confidential information
// Copyright (C) DxO Labs 1999-2013 - (All rights reserved)
// ======================================================================
#include <stdint.h>
#include <stdio.h>
#include "qa/ispHw/fvts/fvtsPlayback.h"
static const SensorCmdSts sensorEventParams_2 = {
        sensorEventBuf_2                // buf
,       78                              // offset
,       50                              // size
};
static const uint8_t sensorEventBuf_4[2] = { 0x00, 0x00 };
static const SensorCmdSts sensorEventParams_4 = {
        sensorEventBuf_4                // buf
,       48                              // offset
,       2                               // size
};
static const uint8_t sensorEventBuf_5[4] = { 0x00, 0x00, 0xc8, 0x00 };
static const SensorCmdSts sensorEventParams_5 = {
        sensorEventBuf_5                // buf
,       68                              // offset
,       4                               // size
};
const uint32_t NbSensorEvents = 3;
const SensorEvent SensorEvents[3] = {
        { 0, EVT_SENSOR_INITIALIZE        , (const SensorEvtData *)NULL                    }
,       { 0, EVT_SENSOR_STATUS_GROUP_OPEN , (const SensorEvtData *)NULL                    }
,       { 0, EVT_SENSOR_STATUS_GET        , (const SensorEvtData *)&sensorEventParams_2    }
};
// ======================================================================
// END OF FILE
// ======================================================================
```

Figure 4: fvtsSensor.c file example

4.5 `fvtsPlayBack.h` declarations

This file declares all functions the chip vendor has to implement. Below is provided the content of this file. It also declares `fvtsEventPlayback()` function which is launcher (i.e. the main function) of a fvts run.

```
// ===========================================================================
// DxO Labs proprietary and confidential information
// Copyright (C) DxO Labs 1999-2013 - (All rights reserved)
// ===========================================================================
void fvtsBench_sendImage(
       const char* filename       // raw name file
,      uint16_t      nbPixX       // number of pixel in X
,      uint16_t      nbPixY       // number of pixel in Y
) ;

void fvtsBench_waitOutput(
       const char* displayName    // display image name to save
,      const char* codecName      // codec image name to save
,      const char* capture0Name   // capture band0 image name to save
,      const char* capture1Name   // capture band1 image name to save
,      const char* faceDetectName // face detection image name to save
) ;

void fvtsBench_setupOutput(
       uint32_t         outputId   // output port number
,      uint8_t          nCfg       // number of expected images
,      uint8_t          nbBands    // number of expected bands
,      const uint16_t* pWidth     // width in pixel for the respective bands
,      uint16_t         height     // height in pixel of the images
,      uint8_t          format     // format of the expected output
,      uint8_t          encoding   // encoding of the exopected output
,      uint8_t          order      // expecting ordering
,      uint8_t          pixWidth   // sample pixwidth
) ;


void fvtsBench_waitInputIt(void) ;


uint32_t* fvtsBench_allocFramebuffer(
       uint8_t          nbFbUsed
,      const uint32_t* pByteSizes
) ;

// ===========================================================================
// END OF FILE
// ===========================================================================
```

Figure 5: fvtsTools.h content

4.6 Test vector executable creation

In addition to test vector C codes and shared items located into the FVTS directory of the delivery, the chip vendor must use three items located into the firmware directory for the bench compilation and linking, described below.

```
firmware/include/DxOISP.h
firmware/objects/ChipVendorPlatform_X/DxOISP_Firmware.o
firmware/objects/ChipVendorPlatform_X/corrData.o
```

DxOISP.h:	File included by ispPlayback_<testName>.c. This is the DxO ISP interface declaration.
DxOISP_Firmware.o:	DxO ISP firmware compiled for the dedicated platform.
corrData.o:	a set of calibration data used at fvts vector generation.

5 CHIP VENDOR BENCH REQUIREMENTS

5.1 Introduction

Verification vectors may be used with chip vendor specific benches:
- Any file to file chip level RTL simulation bench
- Any file to file chip level FPGA bench

Thus, the chip vendor bench must include a host consisting in:
- a mean to monitor the test execution onto the DUT
- a mean to send images to the DxO ISP
- a mean to store images after processing completion from the DxO ISP

As these mandatory features are dependent on the bench architecture and chip vendor's environment, DxO Labs does not deliver these means, but expresses requirements on the functionality and behaviors.

5.2 Required developments

The chip vendor must develop the following functions.

5.2.1 fvtsBench_sendImage()

This function when called into `fvtsPlayback.c` underlines that one image has to be sent to the input of the DxO ISP.

fileName:	file name (relative to the test vector input directory) to provide at the input.
nbPixX:	width of the pixel buffer to provide to the input
nbPixY:	height of the pixel buffer to provide to the input

`fvtsBench_sendImage()` primary responsibility is to set up the input. At the chip vendor choice, it may also start sending the pixels or not. The FVTS environment is robust to both implementations since what really matters is the end of the test, which is ensured by `fvtsBench_waitOutput()`. It is also possible to return from `fvtsBench_sendImage()` only when all pixels have been sent.

This is possible because, in a test, `fvtsBench_setOutput()` are always called before `fvtsBench_sendImage()`. `fvtsBench_sendImage()` is always called before `fvtsBench_waitOutput()`.

This function returns 0 if no problem occurs, else returns 1.

5.2.2 fvtsBench_waitOutput()

This function, when called, must block until the expected output set by a previous call of `fvtsBench_setOutput()` has received the complete expected image(s). Once this is done, the function must save the collected images. These may be checked against references provided through the arguments or let the bit accuracy assertion subsequently to the fvts run.

displayName: file name containing the reference image delivered by DxO for the display output to match against the current collected one. Verification may be done subsequently to the test execution.

codecName: file name containing the reference image delivered by DxO for the video output to match against the current collected one. Verification may be done subsequently to the test execution.

capture0Name: file name containing the reference image delivered by DxO for the capture output first band (i.e. first image) to match against the current collected one. Verification may be done subsequently to the test execution.

Capture1Name: file name containing the reference image delivered by DxO for the capture output second band (i.e. second image) to match against the current collected one. Verification may be done subsequently to the test execution.
Caveat: this is not used for Anapass

faceDetectName: file name containing the reference image delivered by DxO for the faceDetection output to match against the current collected one. Verification may be done subsequently to the test execution.

This function returns 0 if no problem occurs, else returns 1.

5.2.3 fvtsBench_setupOutput()

This function, when called into `fvtsPlayback.c` file underlines that a file is expected at the output (once processing completed) and indicates required information. When the function returns, the output DMA has been properly set and the bench is ready to receive pixels.

outputId:	output port number (display , video , capture , face detection)
nCfg:	number of expected images
nbBands:	number of expected bands (must always be 1 for Anapass)
pWidth:	width in pixel for the respective bands
height:	height in pixel of the images
format:	format of the expected output
encoding:	encoding of the expected output
order:	expecting ordering
pixWidth:	sample pixwidth

This function returns 0 if no problem occurs, else returns 1.

5.2.4 fvtsBench_waitInputIt()

This function must return when the interruption sent by DxO ISP has been received. This is typically a trigger for the wait of the end of a frame.

5.2.5 fvtsBench_allocFrameBuffer()

This function, when called, is in charge of allocating the required memory that is used when use cases needing frame buffers are activated (typically when TNR mode is activated).

nbFbUsed : number of memory segment to allocate (not higher than 1 for Anapass)
pByteSizes : respective size in byte for each memory segment.

This function returns a pointer to 32bit address that will be used subsequently by DxO ISP.

5.3 Test execution

This section put the stress on the real-time aspect of the validation bench. DxO ISP is accessed into the system by two means:

- DxO ISP firmware is accessed by the camera application through procedural calls to configure the DxO ISP (DxOISP_SetCommandStatus() and so on)
- DxO ISP is asynchronously waken up by the IT reception handled by the chip vendor IT handler that's links the IT reception to a call to DxOISP_Event().

Caveat: the FVTS test always asserts preliminary that the DxO entities (hardware, firmware and calibration data) used at execution time are the exact same entites as the ones used at generation time on DxO's site.

This implies a fine management of the asynchronous aspect of the bench depending on what the current purpose is. A FVTS test vector may be executed step by step through GDB over JTAG in order to debug and to validate the process before executing the large amount of tests provided by DxO Labs. The FVTS test vector may be run in a more automatic way to speed the execution.
An exact same test vector will produce two different results (even test problems!) if no IT is handled.

Thus, the chip vendor must ensure two ways of executing the FVTS test vector. A debug mode and an operational mode have to be implemented.

6 WHAT IS A SUCCESSFUL TEST?

A FVTS test vector passes when:

- `fvtsEventPlayback()` function returns 0 (i.e. no assertion has failed and no trouble into chip vendor function bodies occurs)
- all collected and stored images are bit exact with the delivered references