BBC

Research White Paper

WHP 196

May 2011

DVB-T2:
The Common Simulation Platform

Oliver Haffenden

BRITISH BROADCASTING CORPORATION

BBC Research White Paper WHP 196

DVB-T2: The Common Simulation Platform

Oliver Haffenden

Abstract

The DVB-T2 Common Simulation Platform (CSP) is a software model of an end-to-end DVB-T2 chain, begun by SIDSA and AICIA and developed collaboratively by a number of organisations within DVB including Pace and Panasonic. BBC R&D has become a major contributor, making the model fully compliant to the specification; modifying it to participate in the DVB-T2 Validation and Verification work; extending it to work with multiple PLPs and to process T2-MI input; and adding a simulation of the Receiver Buffer Model and features to help with parameter design and checking. This document provides an overview of the CSP and a description of the architecture and principles involved.

The Common Simulation Platform has now been released under an open-source licence on Sourceforge.

Additional key words:

DVB-T2: The Common Simulation Platform

Oliver Haffenden

1 Introduction

The DVBT-2 Common Simulation Platform is a software implementation in MATLAB of the DVB-T2 modulator, demodulator and a channel model. The work was begun by SIDSA, the project was maintained by Vicente Baena of AICIA, and was contributed to Pace and Panasonic as well as BBC R&D, under a collaborative licence agreement. The model has now been released on Sourceforge (http://dvb-t2-csp.sourceforge.net) under the MIT open-source licence.

BBC R&D became one of the largest contributors to the model, and we have been largely responsible for making the CSP compliant to the DVB-T2 standard [1] and for adding almost all of the functionality and modes of the standard including multiple PLPs. We also added the code to generate input streams and test points as required by the Validation and Verification (V&V) process [2], the ability to operate using a T2-MI (Modulator Interface) stream [3] at the input and the option to run the model one frame at a time.

We have also expanded the software suite beyond the original scope, to include a simulation of the receiver buffer model as described in Annex C of the specification; and functions to check the correctness of the configuration and determine the required parameters including design delay and buffer size. Comparison scripts for use in the V&V work are also part of the broader suite.

As a result, BBC R&D is the repository of a great deal of expertise regarding the model, and this document is intended as a record of as much as possible of this knowledge to help others in using and developing the software.

2 Overview of the software

Figure 1 shows an overview of the directory structure.

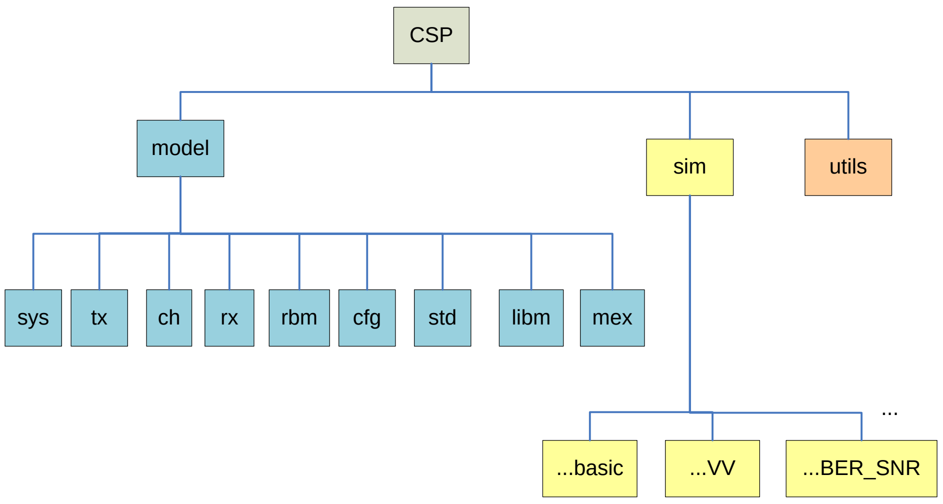


Figure 1: Top-level directory structure

2.1 Top-level directories

The software is divided into three top-level directories:

- "`model`" contains the model of the end-to-end chain including transmitter, channel and receiver, as well as configuration functions. Conceptually this is intended to be used as a library, with the functions being called by the particular simulation from the "sim" directory.
- "`sim`" contains a number of different simulations. These are effectively "main" programs which invoke the model in particular ways appropriate for different applications. For example, there are various types of BER simulation for generating BER curves, a simulation used for generating V&V test points, and another for using T2-MI input.
- "`utils`" contains various utilities including all of the MATLAB scripts used for comparing V&V files from different companies, functions to decode and display L1 signalling and some bash scripts used for running the CSP and comparison scripts overnight.

Strictly speaking the CSP consists of only the "model" directory. However, the model is not enormously useful without the simulation; even if a user intends to develop their own simulation the existing ones provide a good starting point. Furthermore there can be "path" issues if the directories are not arranged as described, and therefore it makes sense to bundle the model and sim directories together.

2.2 The "model" directory

Within the "`model`" directory are the following subdirectories:

- "`tx`" contains the sub-modules making up the transmitter and the transmitter module which calls the other sub-modules in the appropriate order.
- "`rx`" contains the receiver module and its sub-modules.
- "`ch`" contains the channel model
- "`sys`" is an end-to-end T2 system which invokes the transmitter, channel and receiver.
- "`rbm`" contains the receiver buffer model (see section 3.9).
- "`cfg`" contains functions to set up the configuration of the various parts of the chain including the basic independent parameters of the DVB-T2 mode. In a given simulation, these parameters may be overridden later.
- "`std`" contains functions which calculate the dependent parameters of the T2-system according to the DVB-T2 standard. For example, given the FFT size and extended carrier mode flag, the number of active carriers can be calculated. This is typically called shortly after the independent parameters have been finalised, and prevents the need to recalculate the same parameters in many different places in the code.
- "`libm`" contains common functions used by a number of different modules.
- "`mex`" contains compiled versions of functions using the "mex" feature of MATLAB.

2.3 Configuration and "standard" parameters

The modules are configured using a MATLAB structure named `DVBT2`, which is passed to every module and submodule in the chain. `DVBT2` has a very large number of fields and is organised in a hierarchical manner. The following table, adapted from a working document by Vicente Baena, summarises the structure.

Branch	**Structure path**	**Comments**
System	`DVBT2.*`	Variables related to the whole system
Per-PLP	`DVBT2.PLP(*).*`	Variables related to particular PLPs
TX	`DVBT2.TX[.*].*`	Variables to set TX path model behaviour
RX	`DVBT2.RX[.*].*`	Variables to set RX path model behaviour
CH	`DVBT2.CH[.*].*`	Variables to set Channel model behaviour
Simulation	`DVBT2.SIM.*`	Variables to configure test run
Standard	`DVBT2.STANDARD.*`	Dependent variables set according to the standard

The sub-structure `DVBT2.STANDARD` is populated when the relevant "`std`" function is called. It in turn has a hierarchical structure of its own:

Branch	Structure path	Comments
System	`STANDARD.*`	Parameters relating to the T2 system
Inner coding	`STANDARD.PLP(`*n*`).ICOD`	Parameters related to inner (i.e. LDPC) coding
Outer coding	`STANDARD.PLP(n).OCOD`	Parameters related to outer (i.e. BCH) coding

`DVBT2` is passed by value to each of the modules in the signal chain, so modules cannot modify its contents. Any parameters which are dynamic are passed along the chain as part of the signal flow. In particular this applies to the scheduling information, which in some configurations is calculated during processing dependent on the input data. The scheduling information is calculated by the mode adapter and stored in the variable `SCHED` as discussed in section 2.9.

2.4 Signal flow

The input is provided to each module as a MATLAB variable, typically named `DataIn`, and the output is returned in another variable, typically `DataOut`.

2.4.1 Framing for input and output signals

The data in each variable will generally correspond to one or more T2-frames or to one or more Interleaving Frames, depending on the stage in the processing. Through most of the DVB-T2 chain this corresponds directly to one or more units of processing. For example, the BICM chain deals with Interleaving Frames, each of which contains a whole number of FEC blocks. Similarly, the OFDM generation, PAPR processing and guard interval insertion all deal in T2-frames.

However, at the very early stages the signal consists of one or more Transport Streams, and there are not generally a whole number of packets or even bits in an Interleaving Frame or T2-frame. The principle applied at these points in the chain is to output at least enough data for the next block; for the example cited the number of packets generated is rounded up to the next whole number.

Where frame-wise processing is used (see section 2.7), it is the responsibility of the subsequent modules to ensure that any surplus input data that they do not use in a given frame is retained so that it can be used next time.

2.4.2 Data format for input and output signals

The format of `DataIn` and `DataOut` varies from module to module. In general there will be at least one vector or matrix representing the signal for the relevant frame. At many points in the chain there are a number of parallel signal paths. For example, in the stages before the Frame Builder there is a separate path for each PLP; when MISO is used there is a path for each of the two transmitters, and in parts of the receiver there are parallel paths carrying the data and the estimated channel response. The dynamic scheduling information structure `SCHED` has already been mentioned. These parallel paths are implemented by making `DataIn` and `DataOut` into complex data structures. For example, a block might set `DataOut.data` and `DataOut.h_est` for the data and channel estimate paths. `DataOut.data` itself is often a cell array with one cell per PLP containing the output signal for that PLP. In the transmitter, `DataOut.sched` carries scheduling data from one block to the next, and finally `DataOut.l1` carries the L1 data through the transmitter chain.

In the receiver, the CSP currently decodes only one PLP, so the receiver does not have the cell arrays carrying the multiple PLPs. However, a worthwhile addition would be to carry all PLPs through the receiver as well.

2.4.3 Topology

As for most communications systems, a block diagram of a DVB-T2 chain is not in reality a single chain of modules with the output of one feeding the input of the next. There are parallel chains, for example for generating the L1 signalling, which split and come together. Because of the sequential nature of the CSP, the real signal chain needs to be represented as a sequential flow. To achieve this, blocks such as the L1 generation pass on unmodified the signals which they are not concerned with, i.e. the data for the PLPs; similarly the PLP processing stages pass on the L1 data unmodified.

2.4.4 Intermediate file vs memory

The framework can pass signals between modules in two different ways: in memory or using intermediate files.

The former will generally result in increased speed owing to the reduced disc access, whereas the latter allows the intermediate signals to be inspected after the simulation has run, for debugging purposes. Note however that the intermediate file mechanism must be used for some intermediate signals because the relevant signals are used later in the chain. For example, the bit-error ratios (BERs) are calculated by comparing the signal at a particular point in the receiver with the corresponding signal in the transmitter. The mechanism is also used to provide a number of "cheat-wires" to the receiver, for example to allow ideal channel equalisation and to allow the BCH decoder to be emulated by counting the number of bit errors per block.

Where intermediate files are used, the input and output filenames are given by the fields of DVBT2 of the form `DVBT2.<module>.<sub-module>_FDI` and `DVBT2.<module>.<submodule>_FDO` respectively, where `<module>` is `TX`, `RX` or `CH`, and `<submodule>` is the name of the relevant sub-module. For example `DVBT2.TX.MADAPT.FDI` is the input to the Mode Adapter in the Transmitter.

If the relevant field is a null string, then the data is passed between blocks in memory.

In practice, the wrapper framework described in the next section makes this choice (memory or intermediate files) invisible to the implementation of the sub-module itself, since the wrapper takes care of it and passes both the DVBT2 structure and the input data as parameters to the sub-module function.

The convention used for filenames is generally `<submodule>_<module>_do`, where `<submodule>` is the name of the module whose *output* is stored in the file. The input filenames should be set to the output filename for the preceding module. For example, `fadapt_tx_do` is the output of the frame adapter in the transmitter, and happens in most configurations to be the input to the `ofdm` (inverse FFT) module.

Care needs to be taken to ensure that the output of one sub-module is indeed the input of the next, by using the same filename for each or using memory transfer for each. Bizarre bugs often result from getting this wrong. This is a common pitfall when a new module is added or the order of the processing is changed.

2.5 PLP ordering

Multiple PLPs are implemented as follows:

- The `DVBT2` structure contains an array `DVBT2.PLP`, of which each element gives the parameters for the corresponding PLP
- The structure `DVBT2.STANDARD`, containing the dependent parameters arising from the T2 standard, also contains a similar array `DVBT2.STANDARD.PLP`.
- The `DataIn` and `DataOut` variables typically contain cell arrays `DataOut.data` with each cell containing the data for the corresponding PLP

In each case the relevant array or cell array is indexed by what we will term the PLP index. This is an arbitrary index from 1 to `NUM_PLPs` and corresponds to the arbitrary order in which the PLPs are defined. It is distinct from the `PLP_ID`, which can be set to any arbitrary value between 0 and 255. A third concept is the PLP mapping order, which concerns the order in which PLPs are mapped into the frame. It is mandatory to map the common PLPs first, followed by the type 1 PLPs and finally the type 2 PLPs. Within the PLPs of a given type, the CSP will generally map them in order of PLP index.

Most modules will simply deal with PLPs in order of their PLP index, and all parameters are specified in terms of this. The PLP mapping order is only used within the functions that produce the `SCHED` structure, because this makes it easy to perform the relevant arithmetic to generate addresses for the frame mapping. The matrices making up the `SCHED` structure itself are in order of PLP index.

2.6 The transmitter, receiver and channel sub-directories and wrappers

The structures of these three main sub-directories are similar. Taking the transmitter as an example, there is a `tx_blocks` directory, which contains a function known as a *wrapper* for each sub-module in the transmitter chain, and a directory for each concrete implementation of each sub-module.

2.6.1 Wrappers

Rather than call the sub-modules directly, the transmitter module will call the relevant wrapper function. The wrapper then determines which (if any) of several concrete implementations should be called based on the configuration being run.

The concrete implementation is selected according to a field of the `DVBT2` structure of the form `DVBT2.<module>.<submodule>.TYPE`.

As well as selecting between a number of concrete implementations, the wrapper also takes care of the input and output signals as described above, either by reading from or writing to the relevant intermediate file or by accessing a global variable `DATA` used to pass the signals in memory.

There is also a field `DVBT2.<module>.<submodule>.ENABLE`, which determines whether a particular submodule should be run at all. This allows blocks to be skipped entirely.

Note that the effect of disabling a block in this way might not always be meaningful. At the very least, if intermediate files are used for data transfer then the input filename for the subsequent module needs to be changed to match the output of the previous module, since the wrapper will not even write the output file for a disabled module. If memory transfer is used, the value of `DATA` will be left unchanged, which might have the effect of bypassing a module. However, in both cases the resulting data fed to the next module might not be in the correct form for it to process, resulting in an error or, worse, in unpredictable behaviour.

Where a genuine bypass mode is required for a module, such as the constellation rotation in DVB-T2, it has been our practice to make this an explicit parameter of the `DVBT2` structure and to implement the bypass mode explicitly within the module. The `ENABLE` parameter is used only in order that different simulations can use slightly different configurations of the signal chain.

Most of the wrapper functions are all similar to one another, except for the particular fields they access and the particular functions that they call, and in some cases the parameters passed to the functions. It may be possible to collapse them into a single function, perhaps using the feature of function handles (effectively function pointers) offered by MATLAB. This would reduce the work and possibility for mistakes involved in adding new modules.

2.6.2 Concrete implementations

The directories for each concrete implementation contain a file implementing the function that will be called by the wrapper. In general this function accepts the parameters `(DVBT2, FidLogFile, DataIn)` where `DVBT2` is the configuration structure discussed in section 2.3, `FidLogFile` is the file handle to which log messages should be sent, and `DataIn` is the input data for processing. The function returns `DataOut`, the output data to be fed to the next module. The input and output data may represent one or more T2-frames or Interleaving Frames as will be discussed in section 2.4.1. The first module in the modulator (generally `DATAGEN`) is special in that it does not need any input data parameter, although it might read its data from a file instead. Some modules accept additional parameters, such as the transmitted data for some receiver blocks, though this is not entirely consistent.

For many modules, particularly on the transmitter side, only one concrete implementation is provided; after all, there is only one DVB-T2 standard so there is only one correct way of performing many of the operations. Nevertheless, there are some non-prescriptive parts of the standard, for example in relation to the scheduling and allocation model, and so for example there are presently two options for mode adaptation. There is even more potential flexibility on the receiver side since different algorithms could in principle be used; and the channel itself is of course a natural phenomenon and hence a wide range of possible models could be used.

Modules are only called once (or once per frame in frame-wise operation). For modules that need to process multiple PLPs, the PLPs are generally processed in order of PLP Index by an outer loop.

2.7 Frame-wise operation

2.7.1 Background

As originally conceived and implemented, the CSP deals with all the data for a simulation in one go. For example, if four T2-frames were being processed, then first the data generation block would generate enough data for all four frames, then these would be packaged into BBFrames, encoded, modulated, assembled into frames and so on. This made the implementation of the modules more straightforward, since each block would effectively be starting from the beginning of time.

However, the direction taken within the V&V group and beyond eventually made this approach impractical for some applications. Some modes needed to be run for a period of many frames, to simulate fully such features as multi-frame interleaving, frame skipping and FEFs with large FEF intervals. We also wanted to be able to use the CSP to process T2-MI input files of many seconds or even minutes in length. Simulating such long time periods would require MATLAB to hold all of the data in memory, ultimately leading it to run out of memory[1].

2.7.2 Implementation details for frame-wise operation

The decision was eventually taken to modify the CSP to allow it to process one T2-frame at a time. Many stages of the processing are essentially stateless on this scale, since they do the same thing in every T2-frame or are reset every T2-frame. Other blocks are stateless from one Interleaving frame to the next, and this could be accommodated by associating Interleaving Frames with T2-frames. In the transmitter modules, an Interleaving Frame is associated with the first T2-frame to which it is mapped. All of the processing for the Interleaving Frame is performed during the iteration corresponding to that T2 frame. This ensures that the data is ready in time for the later stages which genuinely process T2-frames.

In the receiver, the reverse situation applies, in that the receiver needs to wait until the last T2-frame carrying a particular Interleaving Frame before it can start processing. It then processes the entire Interleaving Frame in one go.

Frame-wise operation is communicated to the model by setting `DVBT2.START_T2_FRAME` and `DVBT2.NUM_SIM_T2_FRAMES`. `START_T2_FRAME` is an index that starts from zero and counts up monotonically for each T2 frame; unlike `FRAME_IDX` it does not reset at superframe boundaries.

`NUM_SIM_T2_FRAMES` is the number of T2-frames to generate at the transmitter output. As is explained below, some modules might need to process more T2-frames' worth of data in order to ensure that they or later modules have enough information to generate the required frames.

`NUM_SIM_T2_FRAMES` is not to be confused with the L1 signalling parameter `NUM_T2_FRAMES`, which indicates the number of T2 frames in one superframe and corresponds to `DVBT2.N_T2` in the CSP.

Generally, `NUM_SIM_T2_FRAMES` is set to 1 for frame-wise operation, but the code is designed to allow more than one T2-frame to be processed at once if desired.

Blocks which work with Interleaving Frames need to know not which T2-frames but which Interleaving Frames they need to process. This is in general different for different PLPs, since the Interleaving Frame can be longer than one T2-frame if P_I and/or I_{jump} are greater than 1.

The information about which Interleaving Frames are to be generated is used in many modules, so they are calculated by the "standard" module. It calculates `START_INT_FRAME` and `NUM_INT_FRAMES` for each PLP, and this can be used in modules without the need to recalculate it. This calculation is only valid for the transmitter modules; for the receiver modules the calculation is different as explained above.

A further complication relates to in-band signalling and to L1-repetition, both of which carry information about future (T2- or Interleaving) frames. The "standard" module works out which T2-frames and Interleaving Frames need to be processed, at least up to the point of allocation to BBFrames, in order to have enough information to generate the L1-signalling and in-band signalling for the T2-frames being generated. These results are stored in `STANDARD.START_T2_FRAME_SIG` and `STANDARD.LAST_T2_FRAME_SIG`. It also calculates `STANDARD.PLP(…).TOTAL_INT_FRAMES_SIG` for each PLP, which indicates the number of Interleaving Frames from the beginning of the simulation need to have been allocated to provide enough signalling information.

[1] The Linux PC on which much of the simulations were run could deal with more frames than the author's Windows XP laptop, though we did not determine to what extent this was a result of having more physical memory, differences in the MATLAB implementation or the Operating System itself. The "crunch point" tended to be the bit interleaver (if running only the transmitter) or the soft demapper, the corresponding point in the receiver, since these points involve coded bits so suffer from the factors of code rate and bits-per constellation.

Fortunately, in most modules the necessary additions have not been too intrusive, thanks mainly to the frame-based processing inherent to many parts of the T2 chain. Many modules need no state at all. However, some modules, particularly the mode adaptation, have suffered somewhat in readability as a result of allowing frame-wise operation.

2.8 State

Different parts of the chain deal with different types of frame, and so at places where the processing frame size changes there will sometimes be more data at the input than is needed for the frames currently being generated. For example, the transition from Interleaving Frames to T2-frames occurs at the Frame Builder, and in the first T2-frame of an Interleaving Frame (for a given PLP) there will be a whole superframe of data at the input, which may be spread over several T2-frames. The Frame Builder is required to store those cells that it does not need for a given T2-frame so that it can use them later. Conversely, in the time de-interleaver in the receiver, the input data might only constitute part of an interleaving frame and therefore the partially received data will need to be stored until the rest of the Interleaving Frame has been received.

The mode adapter is another example where storage is required. In order to generate the in-band signalling for a given Interleaving Frame it needs to process the input data for the following interleaving frame, but this frame is not yet due to output and nor is the in-band signalling data available to be inserted into the BBFrames. The partially processed BBFrames (with space reserved for in-band signalling) need to be stored.

The example of the data generator, which generates entire input packets, was discussed in section 2.4.1.

Unused data that needs to be stored between invocations is the single most common example of state that needs to be maintained.

However, there are other elements of state that need to be stored. For example the total number of bits received and total number of errors need to be maintained by the functions measuring the BER at various points in the receiver.

State is maintained using a global variable `DVBT2_STATE`. The convention is that a given module will use fields of the form `DVBT2_STATE.<submodule>.*` to store its state. This avoids the risk of two modules using the same field name. The pattern in all blocks is that the relevant fields are initialised when the value of `DVBT2.START_T2_FRAME` is equal to zero. The state fields are then read near the beginning of the algorithm and the new state written back near the end.

2.9 The SCHED structure

The process of allocation and scheduling is one of the parts of the DVB-T2 specification that is not completely prescriptive: it can be done in any way desired provided that it is accurately described by the L1 signalling and the signalling within the BBFrames. The CSP supports three different methods for allocation:

- Static allocation. A particular number of BBFrames are allocated to each PLP in each Interleaving Frame
- Allocation model 2. This is described in [4] and [2]. Input data for each PLP is collected during fixed length collection windows. After null packet deletion (NPD) the resulting bits are allocated to BBFrames and padding is used in the last, partially filled BBFrame.
- T2-MI input. In this case the allocation and scheduling is performed by the T2-gateway and the CSP simply reads it from the T2-MI stream.

The allocation process results in a number of FEC blocks for each PLP for each Interleaving frame. The scheduler works out exactly where to put the subslices within the frame, resulting in start addresses for each PLP in each T2-frame, and a subslice interval for each T2-frame.

Because allocation model 2 is data-dependent, the resulting allocation and scheduling information cannot be configured in advance and are calculated in the mode adaptation process. The resulting parameters are stored in a structure called `SCHED`, and this is passed along the chain as a field of the `DataIn` and `DataOut` structures.

The `SCHED` structure contains the following fields (the example is for VV458)

```
             NBLOCKS: {1x7 cell}
        sliceLengths: [7x5 double]
     subsliceLengths: [7x5 double]
      startAddresses: [7x5 double]
   subsliceIntervals: [1560 1560 1560 1560 1560]
```

`NBLOCKS` is a cell array containing a vector for each PLP. The vector gives the number of FEC blocks allocated to that PLP in each interleaving frame.

`subsliceIntervals` gives the value of SUBSLICE_INTERVAL in each T2-frame.

`sliceLengths`, `subsliceLengths` and `startAddresses` give the number of cells in the T2-frame, number of cells in each subslice and start address respectively. There is a row for each PLP and a column for each T2-frame.

Note that all of the fields in the `SCHED` structure begin from the beginning of time, i.e. T2-frame zero, even when `DVBT2.START_T2_FRAME` is greater than zero. It will always contain enough data for the frames being generated this time, so the sizes of the vectors and matrices will increase with time. Starting from the beginning of time makes many of the calculations easier (particularly for multi-frame interleaving and frame-skipping) and does not involve any significant overhead.

2.10 Generation of V&V test points

V&V test points are plain text files used in the Validation and Verification (V&V) process. The files contain sample values at certain points in the DVB-T2 chain. The data is generally divided into "frames" and frames are divided into "blocks". The entities corresponding to frames and blocks depend on the test point: for example in the BICM chain a block is a FEC block and a frame is an Interleaving Frame. In the OFDM processing part, a block is an OFDM symbol and a frame is a T2-frame. In some cases there is no concept of frames and only blocks are present, or vice versa. See [2] for more details of the V&V process.

Test points are inserted by making a call to `write_vv_test_point()`:

```
write_vv_test_point(data, blockLen, blocksPerFrame, tpID, tpFormat,
DVBT2, startBlock, startFrame)
```

Here, `data` is the input data, `blockLen` is the number of samples per block, `blocksPerFrame` is the number of blocks per frame or a vector listing the number of blocks in each frame. `tpName` is the identifier for the test point, including the test point number and the PLP index or transmitter index where relevant. `tpFormat` indicates the data type, since bytes, bits and complex values are formatted slightly differently. `DVBT2` is the ubiquitous configuration structure.

`startBlock` and `startFrame` indicate the index of the first block and first frame in the data being written. This is particularly useful in frame-wise mode, where only a part of the output file is written on each call. The function will start writing the file from the beginning if `startBlock` and `startFrame` are equal to 1, otherwise it assumes that the file has already been started and will append instead.

At some stages in the chain, blocks are of variable size; for example, the P2 symbols from the frame builder have fewer cells than the data symbols. This can be represented by padding with `NaN`s. Any `NaN`s in the data to be written are removed before the block is written, so that the blocks in the output text file vary in size.

Writing V&V test points can be disabled by setting `DVBT2.SIM.EN_VV_FILES=0`. The function `write_vv_test_point` checks the setting of this field, so there is no need to provide an "`if`" statement around the call.

On a fast machine, writing the test point files can be the most time-consuming part of a simulation and disabling them can speed up the simulation dramatically.

3 The sims

As was explained earlier, there are a number of different "simulations" for use in different applications, and writing a new simulation is often the easiest way of expanding the CSP for a new application.

3.1 Common principles

In general, the simulations are in directories named according to the form "`test_dvbt2bl_<sim>`" where <`sim`> describes the particular test case, e.g. "`test_dvbt2bl_basic`". The simulation is invoked from the MATLAB command line using

```
run(Test_Id,Work_Dir_Id,Log_Output, ...)
```

The command line above must be invoked from the "`sim`" directory. The "`run`" function being called here is implemented by "`run.m`" in the sim directory. However, "`run`" is also a built-in MATLAB command, and if the command line is executed from any other location, the built-in "`run`" will be called instead, leading to rather cryptic error messages. The correct version of "`run`" sets up the MATLAB path, generates some filenames and opens some files, and then calls a simulation-specific function called "`test_<Test_Id>.m`".

"`Test_Id`" is the name of the simulation directory without the "`test_`", e.g. "`dvbt2bl_basic`" in the example above. "`Work_Dir_Id`" is the working directory in which the intermediate files will be stored (if used), and "`Log_Output`" is the file descriptor to which log output should be sent.

Subsequent arguments are passed to the particular simulation being run and may be interpreted differently by different simulations. In practice, the simulations that will be presented here all use the following mechanism. The next command line parameter after `Log_Output`, if present, is a cell array containing one or more strings. These strings are MATLAB statements to be executed. This allows the user to override any field of the `DVBT2` structure with a new value. This behaviour is achieved by calling the function override_params(), which both accepts and returns the `DVBT2` structure. Note therefore that attemps to change any other variables will have no effect, since only `DVBT2` is returned.

Common uses for this mechanism include selecting a particular V&V test case, disabling the writing of V&V files, setting directory paths, and skipping blocks.

3.2 "dvbt2bl_basic"

This can be used to perform a basic end-to-end simulation. It calls the relevant configuration function (`t2_cfg_dvbt2blcfg`) to configure the DVBT2 structure, then uses `override_params` to apply any parameter overrides specified on the command line. It then calls the relevant "standard" function (`t2_std_dvbt2blstd`) to set the dependent parameters (see section 2.3). Finally it calls the "system" which actually carries out the simulation. Note that the parameter overriding is done before "standard" is called, so that the calculations and settings are based on the overridden parameters and not the defaults.

This is a very simple simulation which can be used to get started with the CSP. In most practical applications, one of the other simulations is probably more useful.

3.3 “dvbt2bl_VV”

This simulation is used for generating test-point files for the V&V process (see [2]). Like `dvbt2bl_basic`, it starts by calling `t2_cfg_dvbt2blcfg` to set some sensible default configurations. It then calls `override_params`, which in particular allows the user to set `DVBT2.SIM.VV_CONFIG_NAME` to the name of the particular V&V test case to be run. It then calls `set_vv_test_case_params`, which sets up the configuration for the requested test case. This is achieved by first setting the parameters to default values (corresponding to VV001), and then modifying those which differ in a particular test case.

Since this function sets virtually all the parameters, the simulation then calls `override_params` again, so that any overrides that were reversed by `set_vv_test_case_params` can be set again to their desired values. This means, for example, that it is possible to disable the writing of V&V files.

This simulation supports both frame-wise and non-frame-wise operation, according to the setting of the flag `DVBT2.SIM.FRAMEWISE`. In framewise operation, a loop is executed over the configured number of frames. On each iteration, the `DVBT2.START_T2_FRAME` is set to the appropraite value, and `DVBT.NUM_SIM_T2_FRAMES` is always set to 1. The relevant “standard” function is called on each loop iteration, since some of the settings will depend on the frames being generated. The T2 system simulation is then called. In non-framewise operation, the function simply makes one call to “standard” and one call to “system”.

3.4 “dvbt2bl_ber_snr”

This simulation performs BER simulations in various channels. It generates a frame of a T2-like signal, and then runs the channel and receiver a large number of times, using a particular SNR and setting different seed value for the random channel process each time. Once a stopping criterion is reached, it records the BER; if this is above a target value the SNR is increased and the process is repeated until the target is reached. The whole process is enclosed in an outer loop which allows a number of different scenarios to be simulated. Scenarios consist of a particular combination of constellation, code rate and channel parameters and are specified in “`src/cfg/cfg_scenario.m`” within the simulation directory; this also sets the starting SNR value for each scenario.

The signal used by this simulation is only T2-like in that it uses the FFT sizes, constellations and LDPC coding of DVB-T2. However, many other aspects of the standard are not used: in particular there are no pilots at all in the transmitted signal. These are the conditions agreed for simulation of the DVB-T2 system performance and presented in [4]; to determine the expected performance of a real DVB-T2 system it is necessary to add on correction factors as explained in [4].

3.5 “dvbt2bl_ber_snr_dico”

The “`dvbt2bl_ber_snr`” simulation discussed in the previous section is able to give statistically significant simulation results for a target BER by performing long simulations using carefully chosen stopping criteria agreed by the TM-T2 group. However, as a result simulations can take a long time, and it is desirable to start from an appropriate SNR value to avoid spending a long time simulating at too low or too high a BER. Simulations at too low a BER tend to be slow because it takes a very long time to accumulate a statistically significant number of errors. But in DVB-T2, simulations at too high a BER can also take a long time; even though the stopping criterion is reached in a small number of frames, the LDPC decoder will be performing the maximum number of iterations (generally set to 50) on each FEC block in a futile attempt to decode it.

In order to find an appropriate starting point, the `dvbt2bl_ber_snr_dico` simulation should be used. This performs a “dichotomic” search (*búsqueda dicotómica*), starting from widely spaced SNR values guaranteed to bracket the desired BER and performing interval bisection. Only one frame is simulated at each SNR value and so the procedure rapidly homes in on the approximate SNR achieving the target BER.

As for the full BER simulation, the scenarios are specified in "`src/cfg/cfg_scenario.m`" within the simulation directory; this also sets the upper and lower SNR limits.

Note again that the results of this process are approximate; they should not be quoted directly, but only used to provide a starting point for an accurate BER simulation using "`dvbt2bl_ber_snr`".

3.6 "dvbt2bl_ber_snr_VV" and "dvbt2bl_ber_snr_VV_dico"

As was explained in the preceding sections, the standard BER simulations presented in [4] are carried out in a T2-like mode but without pilots and with some other simplifications. The performance of a particular T2 mode can be deduced from this by adding on certain correction factors, but to provide greater confidence, simulations were developed that use fully T2-compliant modes. They do largely the same thing as the corresponding simulation of the preceding sections, except that the DVB-T2 signal is configured according to the V&V test case specified in `DVBT2.SIM.VV_CONFIG_NAME` (see section 3.3).

3.7 "dvbt2bl_T2MIin"

This simulation can be used to operate the CSP with input from a T2-MI file [3]. This can either be a T2-MI stream encapsulated in a Transport Stream using data piping as described in [3], or a "`.t2mi`" file made up of T2-MI packets simply concatenated together. The latter format has become a de-facto standard although it is not specified in [3], and is preferable for this application since undoing the data piping is slow; the CSP runs much more quickly with a `.t2mi` file input.

The simulation works as follows:

- First, the `DVBT2` structure is initialised to some basic defaults using the dvbt2blcfg configuration function.
- `override_params()` is called to allow parameters to be set from the command line.
- `dvbt2micfg` is then called. This performs a first pass over the entire T2-MI file, and sets up the `DVBT2` structure based on both the configurable and dynamic parameters carried in the L1current T2-MI packets. The dynamic parameters are stored in the `NBLOCKS` and `PLP_START` fields of `DVBT2.PLP(*)`, and `DVBT2.SUBSLICE_INTERVAL` is populated with the value for each T2-frame.
- `override_params()` is called again in case any parameters were set to other values by `dvbt2micfg`.
- The model is then invoked in frame-wise mode. For each T2-frame the `START_T2_FRAME` field is set, `dvbt2blstd` is called to set up the dependent parameters and finally the T2 system is called.
- The BBFrames and L1 signalling are read from the T2MI file by the T2-MI version of the data generator (see section 4.1). The T2MI file is closed after each frame but the data generator remembers the file position between frames using the state mechanism (see section 2.8).
- Output is typically written to an IQ file. This is achieved by setting `DVBT2.SIM.OUTPUT_IQ_FILENAME` using the parameter over-riding feature.

3.8 dvbt2bl_VV_parameter_check

This simulation is somewhat different to the others in that it does not actually simulate any T2-Frames at all. Instead, it uses the functions provided to configure the independent and dependent parameters in the `DVBT2` structure, and then calculates both the design delay and the DJB size needed for each PLP.

It reports whether the current settings (if any) are within the safe limits, or suggests values if not.

The calculations are based on the principles described in [4] and are discussed in more detail in a separate White Paper [5].

3.9 “dvbt2bl_rbm” and “dvbt2bl_rbm_T2MIIn”

Annex C of [1] describes a receiver buffer model which should be assumed by the T2-gateway. The CSP includes a MATLAB simulation of the receiver buffer model. It is run separately from the rest of the CSP chain, but uses the `DVBT2` and `SCHED` structures populated by the `CSP`.

There are two different “tests”: `dvbt2bl_rbm` uses intermediate signals from a run of the CSP simulation itself, whereas `dvbt2bl_rbm_T2MIIn` takes input from a T2-MI file. The former method was developed first; whereas the latter method is implemented by converting the T2-MI file data into the same format that the CSP outputs.

The information needed for the receiver buffer model to operate is as follows:

- The Data Field Length (DFL) for each BBFrame. This is stored in a variable and saved in a MATLAB-formatted binary file “`<TestCase>_dflAll.mat`”.
- The Deleted Null Packet (DNP) counts for each packet. This is saved in a text file “`<TestCase>_NullSequence.txt`”
- The configuration parameters for the DVB-T2 mode. This is done by saving the DVBT2 variable in “`<TestCase>_dvbt2.mat`”.
- The scheduling information. This is done by saving the `SCHED` variable in “`<TestCase>_schedule.mat`”

When working with the results of a CSP simulation (`dvbt2bl_rbm`), these files are created by the mode adapter module. They are only created if `DVBT2.SIM.EN_DJB_FILES` is set to “1”.

When T2-MI input is used (`dvbt2bl_rbm_T2MIIn`), there are many similarities with the CSP simulation for T2-MI input described in section 3.7. The `DVBT2` structure is configured in a similar way from the T2-MI file using the same functions; dynamic data is also read using the same functions. The DFL counts and DNP counts are extracted directly from the BBFrames. The `SCHED` structure is also filled using the same function used in the mode adaptation of the CSP.

The receiver buffer model simulation itself is discussed in detail in a separate White Paper [5].

3.10 Block skipping

It can be very time consuming to debug modules that are relatively late in the chain, since it takes a long time for the simulation to reach a point where it goes wrong. The process can be sped up by skipping earlier modules once they have been run once. This is done by setting the “enable” flags for the earlier blocks to “0” (see section 2.6.1). A mechanism was added to most of the simulations to make this easy to set up. The field `DVBT2.SIM.START_AFTER` is set, generally using the parameter overriding function (see section 3.1), specifying which will be the first module to run. A call to `skip_blocks()` in the simulation function then sets the enable flags appropriately. Refer to the `skip_blocks()` function for the values corresponding to each module.

Similarly it can be useful to stop the simulation after a particular block, particularly in frame-wise operation where the simulation will then continue to the next frame. The field `DVBT2.SIM.STOP_AFTER` achieves this effect. The values for this field are the same as for `START_AFTER`.

`START_AFTER` should be used with great care. The first module that is not skipped needs to read its input from the intermediate file generated by its predecessor. Clearly, this will not work if the memory transfer method is used (see section 2.4.4). Also, the preceding modules in the chain must have already been run in order to generate the correct intermediate file *for the correct frame or frames*. Unpredictable and cryptic errors can result if the option is used in frame-wise operation when the previous run simulated more than one frame, leaving a later frame in the intermediate file. Similarly, if the VV test case or any T2 parameters are changed, the intermediate file will contain the wrong data and the full chain will need to be re-run.

Generally, `STOP_AFTER` can be used safely because the CSP topology is entirely feed-forward, so it does not matter that output from earlier blocks is never consumed.

4 The DVB-T2 Transmitter

We will now discuss the modules making up the transmitter model. We will assume a working knowledge of DVB-T2 and concern ourselves principally with aspects particular to the CSP implementation. For a general discussion of the DVB-T2 chain itself, please refer to [4].

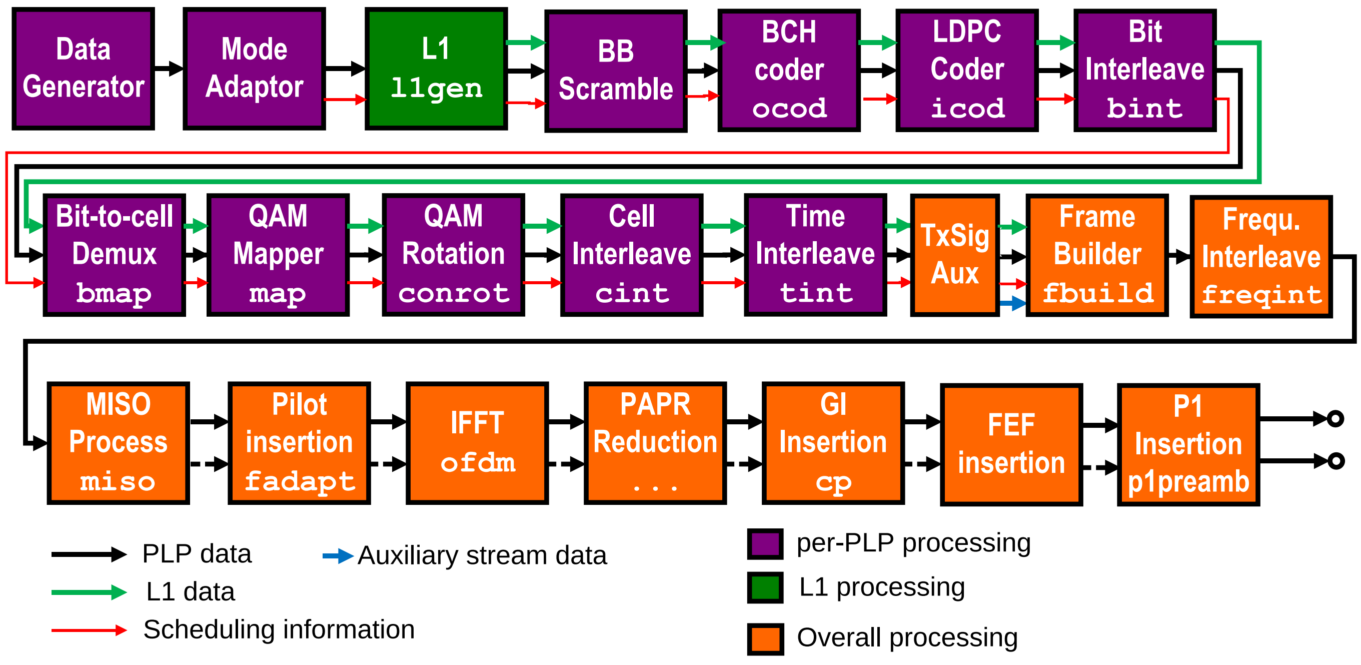


Figure 2: Transmitter signal chain

4.1 Data generator

The data generator module generates the input data for the T2-system. There are the following concrete implementations:

- “`dvbt2bldatagen`” is a random data generator using MATLAB’s built-in random number generator and with no MPEG Transport Stream headers. This is faster to run because it used the built-in functions so is best for use where the input data does not matter, e.g. in BER simulations.
- “`dvbt2itudatagen`” generates an MPEG Transport stream containing the ITU-T O.151 PRBS sequence as used in V&V process [2] for the “non-dynamic” input stream model. Note that this function is slow the first time it runs on a given machine, but it stores the entire sequence in a file, making subsequent invocations much quicker.
- “`dvbt2dynvvdatagen`” generates MPEG Transport Streams made up of a mixture of PRBS packets as described above, SI tables and null packets as used in the V&V process for the dynamic multiple PLP cases. It also performs the TS splitting described in Annex D to generate a common PLP for each group.
- “`dvbt2midatagen`” reads data from a T2-MI file. Since T2-MI contains ready assembled BBFrames, there is no need for the mode adaptation stage when working with T2-MI input, and the signal flow passes directly to the BBFrame scrambling. Since mode adaptation is skipped, this concrete implementation is also responsible for populating and passing on the

SCHED structure (see section 2.9). The information for `SCHED` is taken from the `DVBT2` structure, which is populated during a first pass through the T2-MI file (in “`dvbt2micfg`”).

4.2 Mode adaptor

The mode adaptation block allocates bits to BBFrames and provides the scheduling information via the `SCHED` structure (see section 2.9). It has the following concrete implementations:

- “`dvbt2blmadapt`” implements an allocation model suitable for fixed bit-rate operation. The number of BBFRAMES per Interleaving Frame is pre-defined by the `DVBT2.PLP(*).NBLOCKS` field, and input bits are allocated so as to completely fill the available capacity. This means that the collection windows do not exactly correspond to a fixed period of time, as a result of the deletion of sync bytes and the insertion of other mode adaptation fields.
  For historical reasons, this concrete implementation allows the `NBLOCKS` to be a vector, with each element indicating the number of BBFRAMES in a particular Interleaving Frame. However, this feature should not be used: variable data rate should instead be simulated using null packet deletion and model 2 mode adaptation (next bullet).
- “`dvbt2model2madapt`” implements the dynamic allocation model used in the V&V process; this is based heavily on Allocation Model 2 as described in [4].

4.3 L1 generation

The L1 generation block implements the whole L1 pre- and post-signalling chain, including generation of the bit-fields, arranging the bit-fields into pre- and post- signalling blocks, coding and QAM modulation. From the point of view of the V&V process [2] this represents 13 modules, and consequently there are a 13 test points written by this block.

There are two concrete implementations:

- “`dvbt2bll1gen`” is the normal L1 generator: which generates the L1 fields based on the contents of the `DVBT2` structure. In most cases the signalling fields correspond directly to fields in the `DVBT2` structure and generation is trivial. In other cases certain assumptions have been made. For example, the `STATIC_FLAG` and `STATIC_PADDING_FLAG` are set based on the allocation model and whether the number of blocks allocated is a constant. Some fields in `DVBT2` (such as `FREQUENCY`) are not needed by the model itself and were added solely for the purpose of generating the signalling information. When the specification version is set to 1.2.1 or higher, the module performs bias balancing using the reserved bits and the extension field (if present), according to the algorithm defined in clause 7.2.3.7 of [1].
- “`dvbt2mil1gen`” is the version used when T2-MI file input is being used. The L1 bits are read from the T2-MI file by the “`dvbt2midatagen`” module and passed on as a field of the `DataIn` structure. These L1 bits are processed transparently, allowing the T2-Gateway to make use of the reserved bits and extension field as desired. Note that it is also possible to use “`dvbt2bll1gen`” with T2-MI input, but in this case the bits will be generated only from the fields of the `DVBT2` structure that were populated when the T2-MI file was read. The reserved bits and any extension field will therefore not be taken from the T2-MI input.

4.4 BICM chain modules

The following nine modules implement the BICM chain as described in [1]. These stages are completely prescriptive and therefore only one concrete implementation of each (“`dvbt2bl...`”) is provided. The `SCHED` structure and the coded and modulated L1 cells are passed along this chain from one module to the next.

4.5 Auxiliary stream Tx-SIG generator

This module generates an auxiliary stream containing the Transmitter Signature signal, if used [6]. It is controlled by setting the fields of `DVBT2.AUX`. The output is written into `DataOut.auxData` and the other signals are simply passed straight through.

4.6 Frame builder

This module performs the frame building function described in [1]. This includes:

- Mapping of the L1 pre- and post-signalling cells into the P2 symbols
- Mapping of the common and type 1 PLP cells to the P2 and data symbols
- Mapping the type 2 PLPs to the relevant cells, performing sub-slicing as necessary
- Generation and mapping of the bias balancing cells (if any)
- Generation and mapping of the dummy cells
- Generation and mapping of the "thinning" cells (the unmodulated cells in the frame closing symbol).
- Dividing PLP cells across T2-frames for PLPs using multi-frame interleaving. In frame-wise operation, this involves the storage of cells that have not yet been mapped to a T2-frame, for use in subsequent T2-frames. See section 2.7.

The way in which the PLP cells are mapped is controlled by the `SCHED` structure.

The reader will be aware that different symbols of a frame contain different numbers of data cells, owing to the differing pilot patterns. For ease of implementation, the data at the output of the frame builder is represented by rectangular array in which each symbol has C_{DATA} cells. This is indeed the number of data cells in each data symbol, but the P2 and frame closing symbols have fewer data cells than this. The unused elements at the end of these symbols are set to `NaN`. This also has the effect that the variable-size symbols are written correctly to the V&V output file (see section 2.10.

4.7 OFDM generation

The following modules perform further parts of the specification, and apart from PAPR-TR and FEF insertion need no further discussion here:

- Frequency interleaver
- MISO processing: this performs the conjugation, negation and swapping of the Alamouti processing. The output and subsequent signal chain has two parallel paths corresponding to the two transmitter groups; this is achieved using a cell array with one element for each transmitter group. If MISO is not used, the cell array is still used but has only one element.
- Frame adaptation: this module inserts the pilots and dummy reserved carriers. In MISO mode, the inversion of MISO pilots is also performed in this module.
- OFDM: This performs the inverse FFT.
- PAPR-TR: see below.
- CP: guard interval insertion.
- FEF insertion: see below.
- P1preamb: Inserts the P1 preambles for T2-Frames and FEFs.

4.8 PAPR-TR

This module implements the Tone Reservation method of PAPR reduction (if enabled), including the changes introduced in version 1.2.1 of [1] associated with limiting the amplitude of the reserved carriers and performing the algorithm only on the P2 symbols.

The ACE algorithm has not yet been implemented in the CSP or considered in the V&V work. Implementing it would require information to be carried from the frame builder, through subsequent stages, indicating the PLP or the constellation carried in each cell. This is required so that the algorithm can determine which cells are extendible in a given symbol.

4.9 FEF insertion

The FEF insertion module can generate FEF parts of the "varieties" listed below. The word "variety" is used since the word "type" could be confused with the L1-signalling field FEF_TYPE, which does not necessarily indicate fully what is contained in the FEF part. The varieties supported are:

- PRBS: this is the FEF part defined in [2] for use in the V&V process.
- Null: this FEF part has all zero modulation values (except for the P1 preamble)
- Tx-sig: This contains a Transmitter Signature FEF part as defined by [6].

Note that the P1 preambles are not generated here; they are inserted by the `p1preamb` module.

5 The DVB-T2 receiver

Figure 3 shows the processing chain in the receiver.

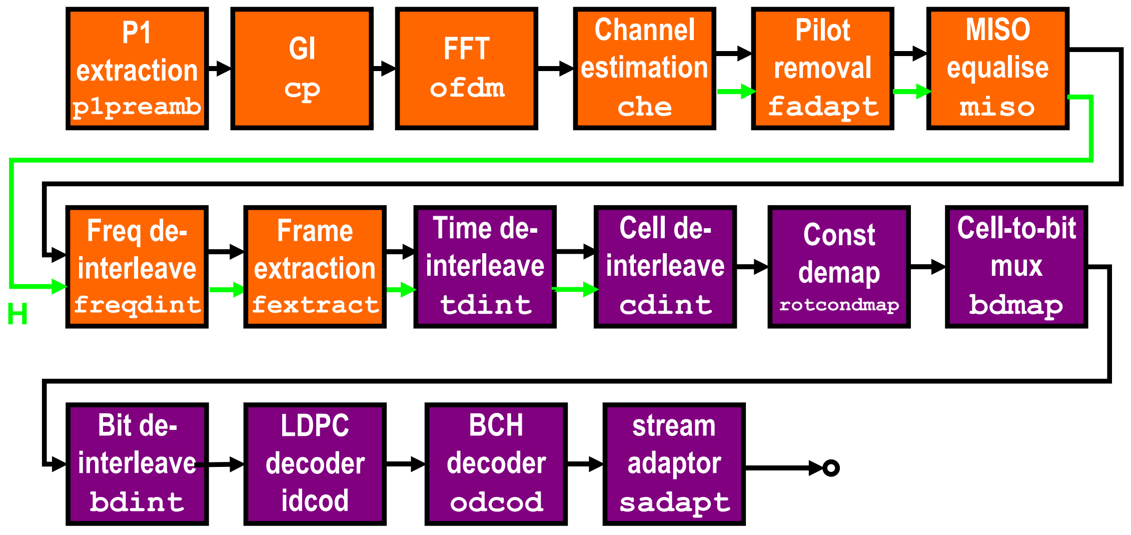


Figure 3: Receiver signal chain

5.1 Assumptions and side-channel information used in the receiver

In order to simplify the processing in the receiver, a number of assumptions are made that would not be valid in general. Essentially the receiver assumes that it is receiving a signal generated by the transmitter part of the CSP itself. It can therefore assume, for example, that the Data Field of the first BBFRAME in the first Interleaving Frame begins with the first bit of a TS packet without needing to read and interpret SYNCD.

The receiver also relies on information provided from intermediate stages in the transmitter; this information would not be available to a real receiver. Specifically, the following signals are used:

- The channel response imposed by the channel model is used by the channel equaliser to perform ideal channel equalisation instead of estimating it from the received pilots
- The channel response is also used to calculate the correct soft decisions, which depend both on the received point and on the signal-to-noise ratio in each cell
- The transmitted bits for each cell are used to perform "genie-aided" soft demapping, which assumes perfect knowledge of the bits in a constellation not currently being decoded [4].
- The transmitted BCH codeword is used to emulate BCH decoding by counting the number of errors in each codeword and correcting them if there are few enough that a full BCH decoder would be able to correct them.
- The transmitted data field lengths (DFL) for each BBFrame are used to extract the data field and discard any padding. This is done in order to prevent catastrophic failure resulting from an error in DFL, causing wanted bits being discarded or unwanted bits retained. The resulting misalignment would cause the BER to go to a half, whereas more meaningful BERs can be obtained.
- The DNP counts, used to re-insert null packets, are taken from the transmitted BBFrames rather than the received ones. This is for a similar reason to the DFLs, i.e. to avoid catastrophic loss of alignment.

5.2 Receiver modules

The receiver comprises the following modules. Most simply implement the reverse of the corresponding transmitter module and will not be discussed further:

- `P1preamb`: Removes and discards the P1 preambles. This module also removes and discards any FEF parts. No attempt is made to decode either.
- Guard-interval removal (`cp`): Removes and discards the guard interval (cyclic prefix)
- `ofdm`: Performs the inverse FFT on the active symbols
- `che`: Channel equaliser – see below
- `fadapt`: removes the pilots and dummy reserved carriers
- `miso`: MISO equalisation – see below
- `freqdint`: frequency de-interleaver
- `l1decode`: L1 decoding – see below
- `fextract`: Frame extraction, the reverse of frame builder. As discussed in 2.4.2, the receiver currently only processes one PLP, so this module only extracts the cells of this one PLP to feed to subsequent modules.
- `tdint`: Time de-interleaver. See section 2.8 for a discussion of state for multi-frame interleaving.
- `cdint`: Cell de-interleaver.
- `rotcondmap`: Calculation of log-likelihood ratios for each bit, including 2D demapping for rotated constellations. See below.
- `bdmap`: re-multiplexing of the bits to reverse the bit-to-cell-word demultiplexer ([1] clause 6.2.1)
- `bdint`: Bit de-interleaver

- `idcod`: LDPC decoder – this is implemented using the decoder in the MATLAB Communications Toolbox
- `odcod`: BCH decoder – this uses emulation as mentioned in section 5.1.
- `sadapt`: stream adaptation. This reverses the stream adaptation, using DFL and DNP values read from the transmitted signal as mentioned in section 5.1.

5.3 Channel estimation and equalisation in SISO and MISO

Channel estimation is performed in the “`che`” block in both SISO and MISO configurations. However, while this module also performs the equalisation in SISO mode, in MISO channel equalisation is delayed until the “`miso`” module, which comes after the frame adapter. This is necessary because equalisation in MISO mode is a process performed on pairs of cells, and these pairs may be broken up by pilots or reserved tones. Only after the pilots have been removed are the resulting data cells reunited in their MISO pairs making the MISO processing straightforward.

5.3.1 Channel estimation

Channel estimation consists of deriving an estimate for each carrier in each symbol of the complex gain from the given frequency domain coefficient (i.e. carrier) in the transmitter to the corresponding coefficient in the receiver.

The channel estimation itself currently has only one concrete implementation, described as “ideal”. There are three different methods used for the estimation process, depending on the channel:

- For channels specified directly as a frequency response, the frequency response can be used directly. The channel is typically specified only at certain frequencies, but the channel equaliser module performs identical interpolation to that done by the channel simulation module itself.
- For static multipath channels with no Doppler, the channel frequency response is calculated directly given the channel parameters.
- For multipath channels with Doppler, the frequency response is determined by loading the transmitted signal after the fading channel has been applied but before noise is added, and dividing it by the clean transmitted signal before the channel is applied.

Note that, as always, the word “ideal” should be interpreted with care. The channel estimation is ideal in the sense that it provides the best estimate of the frequency response at each frequency and on each symbol. In the absence of Doppler and for delays not exceeding the guard interval, this single-tap model is an exact representation of the effect of the channel, because of the properties of Cyclic-Prefix OFDM [7]. However, for longer echoes and for Doppler channels, there will be inter-carrier and possibly inter-symbol interference, so that the channel can no longer be fully characterised by a single complex gain per carrier per symbol.

5.3.2 Channel equalisation in SISO mode

In SISO, the channel equalisation is performed simply by dividing the received complex value in each cell by the complex channel response affecting that cell. This constitutes a Zero Forcing (ZF) equaliser. Other techniques such as Minimum Mean Squared Error (MMSE) equalisation also exist, and are often cited as optimum, but are not yet implemented in the CSP.

5.3.3 Channel equalisation in MISO mode

In MISO, each pair of received cells jointly carries a pair of transmitted constellations, and there are four relevant complex frequency response estimates corresponding to the channel gain from each of the two cells from each of the two transmitters. The MISO equaliser performs the simple matrix inverse operation described in [4]. Again, this represents the ZF solution and it can and has been argued that other solutions, including the MMSE pseudo-inverse, are more optimal.

In SISO mode, the channel response estimate *h* for each cell is carried on through the chain up to the LLR demapper, where it is used to calculate the correct LLR for each bit. In MISO, there are not one but four channel response values affecting each constellation point.

In fact, all that is needed by the LLR demapper is the signal-to-noise ratio for each equalised constellation, and this can be calculated given the four *h* values and the overall SNR. To avoid carrying all four *h*'s through the chain and modifying the LLR demapper, the MISO equaliser instead calculates a single effective h, the value of h that would correspond to the resulting SNR on the equalised constellation. The SNR is not affected by the phase of the channel response and so the effective *h* value can be made real and positive.

5.4 L1 decoding

This module decodes the dynamic L1 signalling and uses it to populate the `SCHED` structure, enabling the frame extraction to locate and extract the desired PLP. The configurable parts of the L1 signalling are assumed to have been written into the `DVBT2` structure.

Two concrete implementations exist:

- "`dvbt2blcheatl1decode`" is a "cheat-wire" decoder. It simply loads the output file from the mode adaptation and extracts the `SCHED` structure from there.
- "`dvbt2blbasicl1decode`" is a basic L1 decoder which decodes the received L1 signalling, as a real receiver would do. It extracts the L1 data cells, performs soft decision decoding and bit de-interleaving and parses the resulting bits to extract the fields of the L1 signalling.

  Currently only the dynamic data is actually made use of: the relevant values are written to the `SCHED` structure, which gets passed along the receiver chain. The configurable parameters are taken from the `DVBT2` structure. Taking the configurable parameters from the L1 signalling would require changes to the architecture to allow the `DVBT2` structure to be modified.

5.5 2D LLR demapping

The "`rotcondmap`" module calculates Log-Likelihood Ratios (LLRs) for each bit in the constellation. Since rotated constellations may be used in DVB-T2, this needs to be a two-dimensional de-mapper, in which the calculation takes into account all of the points of the constellation (i.e. all 256 in 256-QAM), not only the allowed modulation values on each axis.

The module performs the full LLR calculation as described in clause 10.5.3.1.1 of [4].

There is an option to perform "Genie-Aided" (GA) demapping as described in clause 10.5.3.2.2 of [4]. Genie-aided demapping requires a knowledge of the transmitted bits, which is achieved by loading the relevant intermediate file from the transmitter.

The cheating involved in GA demapping is not quite as breathtakingly blatant as it sounds. Knowledge only of the *other* bits carried in the same constellation, not of the bit currently being decoded, is used in the calculation. This is intended to approximate the limiting performance of Iterative Demapping (clause 10.5.3.2.1 of [4]), in which information about those other bits is indeed available from the LDPC decoder.

6 V&V scripts

A set of scripts have been developed at BBC R&D to support the V&V process [2]. These currently run on a server machine running Debian Linux and installed in the server room at BBC R&D's South Lab in west London. The V&V files from the companies participating are stored on an FTP site hosted by Screen Service Broadcasting Technologies (SSBT).

6.1 Comparison scripts

These do the job of comparing the V&V test files generated by the various companies taking part in the V&V exercise, including the files generated and uploaded by the CSP (section 6.2). Typically they are run nightly as a cron job.

The main script is `VV_compare.bash` in the `utils` directory of the CSP suite. This downloads any new test files from the V&V FTP site, compares all the files and generates a set of html reports which it uploads back to the FTP site.

The implementation finally arrived at is slightly involved:

1. A local mirror copy of the FTP site is kept in a directory `/home/dvbt2/VV/master` on the server machine. The first stage is to synchronise this with the remote FTP server, using `lftp` to download only files that are more recent on the remove server. To traverse the R&D firewall, `lftp` is run under `tsocks`.
2. The entire `/home/dvbt2/VV/master` directory is copied to a new directory `/home/dvbt2/VV/working`, after first deleting the latter directory if it exists. This has the effect of removing any vestigial extracted files that would otherwise persist.
3. The working directory is traversed and any compressed archives are extracted in-place. The convention in uploading compressed files to the FTP server is that they should yield testpoint files in the correct hierarchy if extracted in this way.
4. The sub-directories in the `TopDown` directory are examined; most should represent a test case. The script uses a wildcard to select the desired test cases. The wildcard can be provided on the command line; otherwise a default wildcard of “`VV*`” is used, matching all of the test cases that use the current naming convention.
5. For each test case identified in the preceding step, MATLAB is invoked with the function “`MakeVVReport`” and the appropriate parameters. This produces two temporary output files, a detailed report for the test-case and a short summary for inclusion in a single overall summary report for all test cases.
6. The detailed report is made up of html elements; the script adds a header and footer to it to make it syntactically correct and to allow styles to be imposed easily. It then uploads the resulting report to the FTP site.
7. The summary report also contains html elements. These are appended to the overall summary file; a header and title for this report are generated at the beginning of the run. The filename of the summary report is based on the wildcard provided. This prevents a large summary report containing many test cases from being overwritten by a very short report with only a few test cases.
8. Steps 5-7 are repeated for each test case.
9. The summary report is finished off by appending the footer, and uploaded to the FTP site.

The reason for copying the whole “master” directory to the “working” directory is that it allows the FTP mirroring to work on a directory that is not touched by the comparison process. If the zip files were expanded in the “master” directory, this would make “master” different from the remote FTP directory and confuse the mirror process. Another problem would arise if an archive were uploaded that did not contain test points that had been present in a previous version: those extracted files would remain in place, potentially causing a misleading mismatch.

No doubt a better scheme could be devised, perhaps using a makefile to determine which files needed to be extracted rather than extracting every file every time. However, the existing scheme seems to work well enough. The copying and extraction only takes a small proportion of the total execution time.

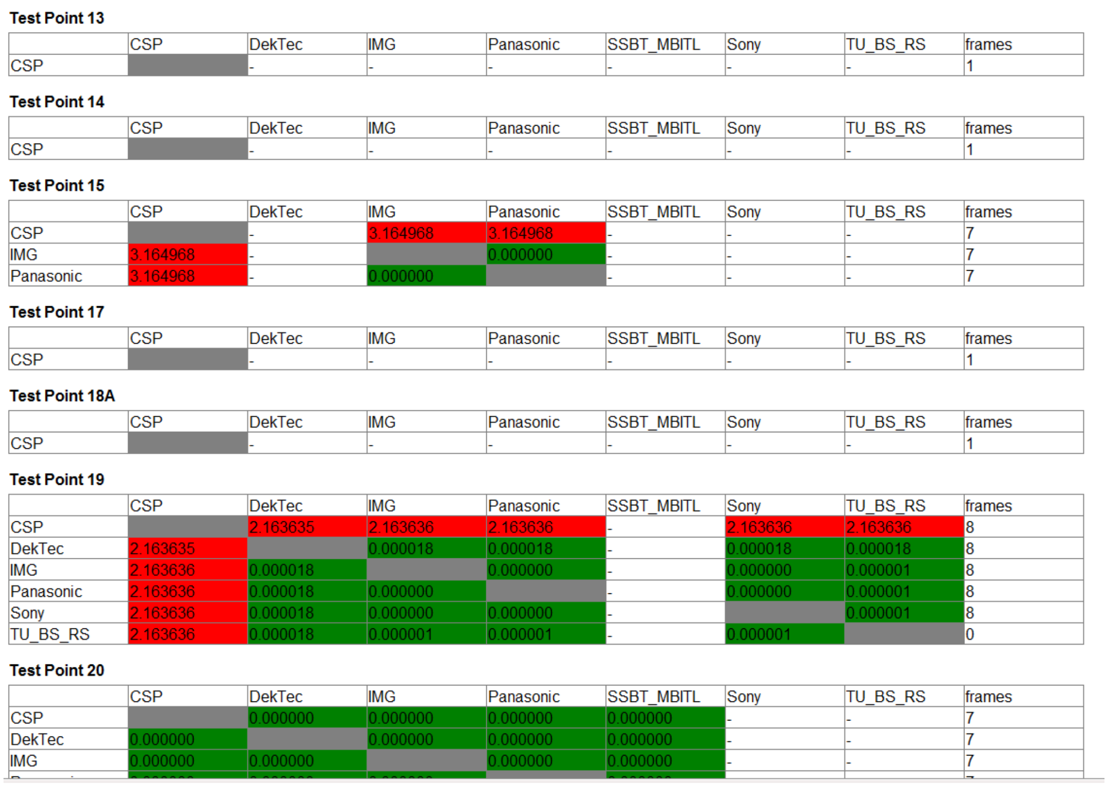
6.2 MakeVVReport

`MakeVVReport`() is the function that generates both the report on an individual test case and the section of the summary report related to that test case.

- The function will compare all test points it finds unless an explicit list is given as a parameter.
- For each test point, the files are compared a “block” at a time. This minimises the memory usage and makes it easier to deal with blocks of varying sizes.
- The data type (bytes, bits, complex) for each test point affects the way in which the comparison is performed. The function `CompareVVPointBlockwise()` includes a lookup table giving the type for each test point. If a test point does not appear, the scripts will attempt to deduce it automatically, but this process is not completely failsafe and it is better to add any new test points to the list. Ideally the data type would be indicated in the header of the test file, as some companies already do.
- Files for a given test-point are compared pair-wise, i.e. each company’s file is compared with each other company.
- By default, complex test-points are normalised to the same rms value before taking the difference. This allows for differing scale factors, particularly for fixed-point implementations.
- Binary and byte test points are not normalised.

6.3 Individual reports

A report is generated for each test case. Figure 4 shows part of a typical report.



Test Point 13

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		-	-	-	-	-	-	1

Test Point 14

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		-	-	-	-	-	-	1

Test Point 15

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		-	3.164968	3.164968	-	-	-	7
IMG	3.164968	-		0.000000	-	-	-	7
Panasonic	3.164968	-	0.000000		-	-	-	7

Test Point 17

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		-	-	-	-	-	-	1

Test Point 18A

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		-	-	-	-	-	-	1

Test Point 19

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		2.163635	2.163636	2.163636	-	2.163636	2.163636	8
DekTec	2.163635		0.000018	0.000018	-	0.000018	0.000018	8
IMG	2.163636	0.000018		0.000000	-	0.000000	0.000001	8
Panasonic	2.163636	0.000018	0.000000		-	0.000000	0.000001	8
Sony	2.163636	0.000018	0.000000	0.000000	-		0.000001	8
TU_BS_RS	2.163636	0.000018	0.000001	0.000001	-	0.000001		0

Test Point 20

	CSP	DekTec	IMG	Panasonic	SSBT_MBITL	Sony	TU_BS_RS	frames
CSP		0.000000	0.000000	0.000000	0.000000	-	-	7
DekTec	0.000000		0.000000	0.000000	0.000000	-	-	7
IMG	0.000000	0.000000		0.000000	0.000000	-	-	7

Figure 4: Extract of a typical test case report

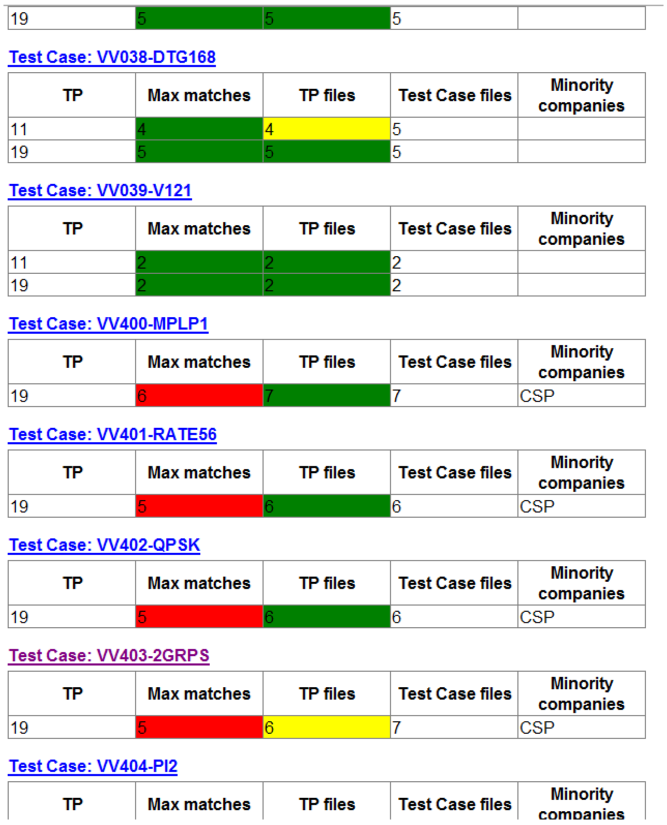
The result for each test point is presented as a table with one row for each company that provided that test point, and one column for each company that provided any test points for this case. This means that the columns for a particular company will always be in the same horizontal position throughout a given report. The entries in the table give the maximum difference between the files of the two companies concerned.

The table entries are coloured green if the worst difference is less than a threshold value and red otherwise. The threshold is currently set to -30dB, although in practice implementations now match more closely than this. One situation in which the worst-case mismatch can give a pessimistic impression is when a large sample is clipped in one implementation and not in another.

The report also gives the number of frames that were compared for each implementation. Not all implementations indicate the frame boundaries, so this can sometimes read “0”. The number of frames should be noted when interpreting reports. For example, if company A provides only one frame, whilst companies B and C provide four frames, the situation can arise where A matches both B and C, but B does not match C, because the mismatch happens in a later frame.

6.3.1 The summary report

Figure 5 is an extract of a typical summary report.



19	🟩 5	🟩 5	5	

Test Case: VV038-DTG168

TP	Max matches	TP files	Test Case files	Minority companies
11	🟩 4	🟨 4	5	
19	🟩 5	🟩 5	5	

Test Case: VV039-V121

TP	Max matches	TP files	Test Case files	Minority companies
11	🟩 2	🟩 2	2	
19	🟩 2	🟩 2	2	

Test Case: VV400-MPLP1

TP	Max matches	TP files	Test Case files	Minority companies
19	🟥 6	🟩 7	7	CSP

Test Case: VV401-RATE56

TP	Max matches	TP files	Test Case files	Minority companies
19	🟥 5	🟩 6	6	CSP

Test Case: VV402-QPSK

TP	Max matches	TP files	Test Case files	Minority companies
19	🟥 5	🟩 6	6	CSP

Test Case: VV403-2GRPS

TP	Max matches	TP files	Test Case files	Minority companies
19	🟥 5	🟨 6	7	CSP

Test Case: VV404-PI2

TP	Max matches	TP files	Test Case files	Minority companies

Figure 5: Extract of a typical summary report

The summary report gives results for all the test cases compared in a particular run, but for each one it includes only certain test points, typically TP11 (the end of the BICM chain) and TP19 (the final I/Q output). For each point, only the maximum number of matching implementations is given, together with a list of “minority” companies. Here, “matching” means that the worst-case error is less than the threshold (currently -30dB; see above).

For example, if A,B and C match each other, D and E match each other and F does not match anybody, the maximum number of matches is three, whilst D, E and F are the “minority” implementations.

“Minority” does not necessarily mean “wrong” – minority companies may have implemented something correctly where the other companies have all made the same mistake, or updated to new parameters before the others have got round to it (this is the situation in Figure 5).

The maximum number of matches will be shaded red if there are any implementations that do not match. The column “TP files” gives the number of files provided for the test point in question; this will be shaded yellow if not all companies contributing to the test case had provided that test point. "Test Case files" indicates the number of companies contributing to the test case as a whole.

6.4 Overnight CSP execution

The bash script `VV_run_CSP.bash` in the `utils` directory can be used to generate one or more V&V test cases and zip and upload the resulting test files to the V&V site. The script is intended to run on the server PC, from a "working copy" of the Subversion repository containing the CSP suite, and is most often run overnight as a "cron" job. For details of the repository see section 10.

The script first checks whether there have been any changes in the repository. This check is achieved by determining the current revision of the working copy (using "`svn info`"), performing an "`svn up`" operation to update the working copy, and then checking whether the revision has changed as a result. If there have been no changes, then it does not need to run the CSP this time. The CSP can be forced to run by executing `./rerun` in the `utils` directory. This touches a file `re.run` in the CSPData directory; if the file exists then the CSP runs whether or not the repository has changed (it still does the update anyway).

It then works through a list of V&V configurations, launching MATLAB and running the "`test_dvbt2bl_VV`" simulation (see section 3.3) for each one. The list can be provided on the command line; if no command-line arguments are provided then the script will read the list from a text file `CSPConfigurations.txt` in the "`sim`" directory. This file is itself under version control, so can be edited in another working copy and committed to the repository, causing the new list of configurations to be run.

The resulting test point files for each configuration are then zipped and uploaded to the V&V FTP site. Before doing the FTP transfer, the script checks whether there have been any significant changes. This is done by concatenating all of the output files, discarding header and comment lines, and calculating an MD5 hash. The MD5 hash is written to a file; only if the contents have changed compared to the previous file is the FTP transfer performed.

Because the FTP server is remote from the server running the scripts, the FTP transfer does not always succeed. If the script detects a failure it touches the `re.run` file (see above) ensuring that the whole CSP script will be executed again the following day.

To ensure that the files on the FTP site are updated regularly, the scripts are forced to run and upload files every Friday night by an additional cron job, which touches the `re.run` file and also deletes all the MD5 files.

One method of working is for developers to work on their own local working copy, checking in ("committing") changes to the repository, which will then be pulled onto the server machine when it runs the cron job. On days when no changes have been made, the CSP will not be run and no new files will be uploaded.

6.5 CSP script for T2-MI operation

The script "`T2MI_run_csp.bash`" is not really part of the V&V process. It can be used for generating a set of IQ files from input T2-MI files. The list of test cases to be generated is provided on the command line.

7 Conclusions

The Common Simulation Platform is a comprehensive software model of a DVB-T2 end-to-end chain. Following extensive contributions by BBC R&D as well as the work done by other contributors, it now supports almost all options of the DVB-T2 specification. Our participation in the V&V exercise has shown it to be compliant to the DVB-T2 specification.

We have also added other functionality, including automatic comparison scripts used extensively for the V&V work, a simulation of the receiver buffer model of the T2 specification, tools for checking parameters and configurations for reading T2-MI files and generating output IQ files.

In February 2011 the CSP was agreed to be in a state where it could be released publicly as a reference model and was placed on the Sourceforge site from where it can be freely downloaded (see section 10 below for details).

8 Acknowledgements

The CSP was originally developed by SIDSA and has been maintained and extended by Vicente Baena and his colleagues at AICIA. Vicente also performed careful checking and correction of our contributions prior to releasing them. There were also contributions from Caleb Price of Pace Microelectronics and Mihail Petrov of Panasonic. Ignacio Lacadena of SIDSA chaired the Simulation task-force in which the CSP development was co-ordinated. Zhong Yi (Oscar) Hu of SSBT managed the FTP site used for the V&V exercise. The V&V group was chaired by Lachlan Michael of Sony.

Andrew Murphy and Chris Nokes made major contributions to the CSP development at BBC R&D. Andrew took care of much of the difficult scripting for the V&V comparison. The parameter checking simulation was Chris's work. Peter Moss provided the implementation of Tone Reservation on which the CSP module was based.

9 References

[1] ETSI EN 302 755, "Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2), V1.1.1, September 2009

[2] DVB, "The DVB-T2 Reference Streams", September 2010. (Available at http://www.dvb.org/technology/dvbt2/DVB-T2_ReferenceStream-Documentation.pdf or ftp://ftp.kw.bbc.co.uk/t2refs)

[3] ETSI TS 102 773, "Digital Video Broadcasting (DVB); Modulator Interface (T2-MI) for a second generation digital terrestrial television broadcasting system (DVB-T2), V1.1.1, September 2009

[4] ETSI TS 102 831, "Digital Video Broadcasting (DVB); Implementation guidelines for a second generation digital terrestrial television broadcasting system (DVB-T2)", October 2010

[5] Nokes, C.R. and Haffenden, O.P., "DVBT2: The Receiver Buffer Model", BBC R&D White Paper. *Publication pending*

[6] ETSI TS 102 992 / DVB Blue Book A150, "Digital Video Broadcasting (DVB); Structure and modulation of optional transmitter signatures (T2-TX-SIG) for use with the DVB-T2 second generation digital terrestrial television broadcasting system", September 2010

[7] Stott, J.H., "The how and why of COFDM", EBU Technical Review, Winter 1999

10 Code location

The code for the CSP can be downloaded from http://dvb-t2-csp.sourceforge.net.

Release versions will be made available to download from time to time.

The code is held in a Subversion repository – see the "Develop" tab on the project's Sourceforge page for details. It is likely that the CSP will undergo further development, for example to include future extensions of DVB-T2 and to correct any bugs that are identified.

The very latest code can always be obtained by checking out the "head" revision from the repository using a suitable Subversion client, e.g. Tortoise SVN (http://tortoisesvn.tigris.org/), though it should be noted that this version is likely to be less well tested and less stable than the release version.

Use and distribution of the code is subject to the MIT licence as detailed in the headers of each file.