DVB-TX-IRIS

0.1.1

Generated by Doxygen 1.8.6

Fri Oct 28 2016 13:01:11

Contents

Chapter 1

DVB-TX-IRIS

1.1 Introduction

This package contains the components which may be used to create DVB-T radios with the IRIS software radio framework.

1.1.1 The DVB-T waveform

DVB-T uses orthogonal frequency division multiplexing (OFDM) symbols with cyclic prefix in order to deliver the transmitted data over the communication channel. OFDM symbols are grouped in frames (composed of $N_F = 68$ OFDM symbols) and superframes (composed of 4 frames): the superframe can be considered to represent a basic group of data, as it always carries an integer number of transport stream (TS) packets, which constitute the payload of DVB-T and carry compressed video and audio streams. The base-band (BB) signal samples can be expressed as

$$\tilde{s}[n] = \sum_{m=0}^{+\infty} \sum_{l=0}^{N_F-1} z_{m,l}[n] = \sum_{m=0}^{+\infty} \sum_{l=0}^{N_F-1} \sum_{k=0}^{K-1} c_{m,l,k} G_k \psi_{m,l,k}[n]$$

where m represents the frame index, l is the OFDM symbol index, k is the subcarrier index, K is the number of active carriers (depending on the transmission mode), and N_F is the number of OFDM symbols per frame; the data transported over each carrier is given by $c_{m,l,k}$ and it is a QAM (quadrature amplitude modulation) mapped constellation symbol, carrying ν bits per symbol; G_k is a carrier amplitude weighting factor that can be used to precompensate linear distortions introduced by the transmitter ($G_k = 1$ in case of no distortions), and $z_{m,l}[n]$ is the OFDM symbol in time. The modulation is performed using K out of NFFT orthogonal carriers $\psi_{m,l,k}[n]$, expressed as

$$\psi_{m,l,k}[n] = e^{j2\pi \frac{k-K_2}{N_{\text{FFT}}}(n-N_G-(l+mN_F)N_S)} \cdot \Pi_{N_S}[n-(l+mN_F)N_S]$$

where $K_2 = K/2$, N_G is the number of samples of the guard interval, $N_S = N_{\text{FFT}} + N_G$ is the total number of samples of the OFDM symbol, and $\Pi_{N_s}[n]$ is the boxcar window, which is equal to 1 in $[0, N_S - 1]$ and to 0 elsewhere. The BB samples are then converted into the analog domain using a sample time $T_{s,\text{DVBT}}$ that depends on the bandwidth of the DVB-T configuration. The sample rate $f_{s,\text{DVBT}} = 1/T_{s,\text{DVBT}}$, can be replaced by the DAC sample rate $f_{s,\text{DAC}} = 1/T_{s,\text{DAC}}$, as expressed by

$$\tilde{s}(t) = \sum_{n=0}^{\infty} \tilde{s}[n]\, h(t - nT_{s,\text{DVBT}}) = \sum_{n=0}^{\infty} \tilde{y}[n]\, h_I(t - nT_{s,\text{DAC}})$$

where $h(t) = T_{s,\text{DVBT}} \text{sinc}(\Pi t/T_{s,\text{DVBT}})$ is the ideal BB reconstruction filter, $h_I(t)$ is the DAC output filter, and $\tilde{y}[n]$ is the signal $\tilde{s}[n]$ resampled to the DAC sample rate. Eventually, the analog signal is up-converted, using a quadrature modulator, to the RF carrier frequency, f_c, as

$$s(t) = \mathrm{Re}\left\{\tilde{s}(t)\, e^{j2\pi f_c t}\right\}$$

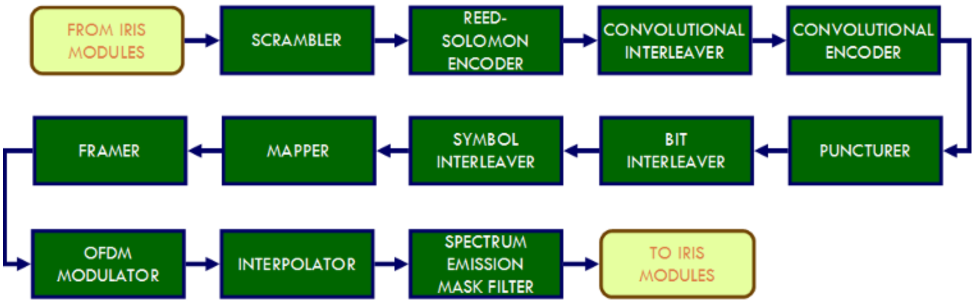


Figure 1.1: DVB-T transmission scheme.

1.1.2 IRIS

The DVB-TX-IRIS extension is an extension of the Iris framework. Iris is a software architecture for building highly reconfigurable radio networks using a component-based design. The architecture comprises two repositories - Iris_Core and Iris_Modules. Iris_Core contains the core part of the architecture such as parsers, managers, and engines. Iris_Modules contain the components which can be used to create a software radio such as PHY-layer components and radio controllers. The Iris architecture, written in C++, supports all layers of the network stack and provides a platform for the development of not only reconfigurable point-to-point radio links, but complete networks of reconfigurable radios. Individual radios are described using an XML document. This lists the components which comprise the radio, gives the values to be used for their parameters and describes the connections between them. Iris was originally developed by CTVR, The Telecommunications Research Centre, based at University of Dublin, Trinity College. In 2013, it was released under the LGPL v3 license and is currently managed by Software Radio Systems (http://www.softwareradiosystems.com).

Since DVB-TX-IRIS extends Iris functionalities, there are shared requirements that should be satisfied from the software point of view. In particular, they are:

- Essential SW
  - Ubuntu Linux OS 32/64 bit (http://www.ubuntu.org), release 14.04 or later
  - CMake 2.6 or later (http://www.cmake.org/), an automated software build and test environment for C/C++
  - Boost 1.46 or later (http://www.boost.org/), an extensive collection of C++ libraries for accelerating common software tasks
  - Iris_Core (http://www.hostedredmine.com/projects/iris_software_radio/wiki), the core system of the Iris framework
  - Iris_Modules (http://www.hostedredmine.com/projects/iris_software_radio/wiki), additional modules for the Iris framework
  - FFTW (http://www.fftw.org/), a powerful C/C++ library for FFT transforms
  - UHD (http://code.ettus.com/redmine/ettus/projects/uhd/wiki), needed for the connection to USRP hardware
- Optional SW
  - Qt 4.8 (http://qt-project.org/), used for building graphical widgets
  - Qwt 6 (http://qwt.sourceforge.net/), used for building graphical widgets
  - Liquid-DSP (https://github.com/jgaeddert/liquid-dsp), for some PHY components

  - Google Protocol Buffers (https://developers.google.com/protocol-buffers/), for some Stack components
  - Python (http://www.python.org/), for the PythonPlotter widget
  - Octave (http://www.octave.org/), for recreating the test vectors used during the testing phase of the build, and for running complete TX/RX simulations
  - Matlab (http://www.mathworks.com/), for the MatlabTemplate PHY component and MatlabPlotter widget
  - Doxygen (http://www.doxygen.org/), for the documentation
  - tzap (dvb-apps package) and w_scan (w-scan package), used for real-time stream quality testing with DVB-T USB receivers

From the hardware point of view, the following items are required:

- Essential HW
  - A workstation or laptop PC equipped with a multicore CPU clocked at 2 GHz or more, 4 GB of RAM, 20 GB of free disk space, and a free Gigabit Ethernet connection
  - An Ettus USRP N210 equipped with an UHF/VHF capable daughterboard (such as the SBX or SBX120)
  - A UHF/VHF antenna (preferably directional antenna for longer communication range)
  - A DVB-T capable receiver (such as a TV set, a set-top box, or an USB dongle, provided with indoor reception antenna)
- Optional HW
  - A spectrum analyzer for verifying the spectrum of the emitted DVB-T signal

1.2 Compilation and installation

Provided that the essential SW requirements are satisfied, the following steps are needed to successfully compile DVB-TX-IRIS:

• The DVB-TX-IRIS code can be downloaded using SVN or GIT from the official GitHub page of the WiSHFUL project, https://github.com/wishful-project. For instance, using subversion, the command is

```
svn checkout http://url.iris.repository/dti_wishful
```

• Now cd into the project directory, and create a folder named "build", where the extension will be compiled

```
cd dti_wishful
mkdir build
cd build
```

• Invoke cmake to prepare the build environment

```
cmake ..
```

• Invoke make to build the extension

```
make
```

• Optionally, you can benchmark the speed of the extension components and test their correct operation with

```
make benchmark
make test
```

• If the previous steps ended with success, now you can install the extension by typing

```
sudo make install
```

• If you want to build also the documentation HTML manual, then you must execute doxygen in the "doc" folder

```
cd ..
cd doc
doxygen
```

1.3 Choosing a bit rate and the transmission parameters

The DVB-T system is designed to convey a constant bit rate payload. Thus, if the video and audio sources, after compression, have a bit rate higher or lower than the expected value, the SDR modulator will fail in delivering an high quality stream to the receivers. The connection among bit rates and transmission parameters is shown in Table III. There, for any combination of modulation, cyclic prefix, and code rate, there is a corresponding value that should be used to generate the TS. We also highlight that the OFDM mode (2K, 8K) does not influence the bit rate. Summarizing, the transmission parameters could be chosen according to the following guidelines:

- Cyclic prefix: this parameter determines the ruggedness of the DVB-T signal with respect to RF channel impairments such as multipath and echoes. Choose a large value (1/4) if delayed signal echoes are expected, especially when covering large areas (width on the order of kms). Differently, the smallest value (1/32) is sufficient for a localized transmission covering a range of few hundreds of metres.
- OFDM mode: as explained before, this parameter does not impact on the payload bit rate. However, a 2K OFDM mode is to be preferred if the receivers are preferentially of nomadic or mobile type, since in this case the harmful effect of the Doppler spread (due to the relative motion of the transmitter and receiver) on OFDM carriers orthogonality is minimized. Moreover, 2K is preferred for covering relatively small areas. On the other side, 8K can be used when covering large areas and the receivers are expected to be of a static, fixed type.
- Modulation order: this parameter selects the spectral efficiency of the system, thus larger modulation orders (64-QAM) will allow higher bit rate payloads to be delivered. Concurrently, higher modulation orders also require important values of S/N ratio at the receivers, in order to have a successful reception of the TV signal. On the other side, smaller modulation orders (4-QAM, 16-QAM) have lower requests in terms of S/N ratio, but they are not as spectrally efficient.
- Code rate: this is the other parameter that concurs to determine the spectral efficiency of the system. High values of code rate (e.g., 7/8) allow very high bit rate payloads, but protect less the signal from unwanted interferences. Lower code rates (such as 1/2) protect the signal very well from noise and other disruptive impairments, but they also lower the spectral efficiency.

We conclude this subsection by recalling that, anyway, the choice of the bit rate depends also on the quality and quantity of the TV programs that will be included in the multiplexed TS.

1.4 Choosing frequency, power, gain

These three parameters do not concur in the modification of the payload bit rate, but they are important as well.

- Frequency: the emission frequency should be chosen in the range that is commonly used by TV receivers in the VHF-UHF bands. Choosing an emission frequency outside of this range could result in the impossibility to receive this signal on common TV receivers. However, it should be noted that an official permit of the National or Regional Communications Authority must be accorded before broadcasting at such frequencies. The relationship between a DVB-T UHF frequency channel i and its central frequency (for 8 MHz systems) is

  $$474 + i \times 8MHz,\ i = 0, 1, 2, \ldots$$

- Power: this parameter decides the power of the digital signal generated by the SDR DVB-T modulator. This value can be overridden by applying proper amplification gains in the transmission chain, nonetheless it should be chosen carefully. In fact, a typical value of 30-50 (%) is recommended, since this will result in a digital signal with a smaller dynamic range, which will produce fewer distortions during the D/A conversion stage (i.e., less clipping noise).

- Gain: this parameter is related to the analog amplification stage of the USRP. Smaller values (0-5 dB) will result in a scarcely amplified signal, that will not cover a large area, but it will be less distorted since the amplifier is working in the linear portion of its amplification characteristic curve. Higher values (15-20 dB) will allow covering larger areas at the expense of the spectral flatness and purity of the emitted RF signal. The highest values should be avoided, since the additional emitted power is obtained by operating in a highly nonlinear portion of the amplifier characteristic, degrading completely the main spectrum and its surrounding frequencies.

1.5 Choosing an input TS file

Once the transmission configuration has been selected, the input TS can be chosen among one of those that are already provided for this purpose, and that can be downloaded from

http://dante.diei.unipg.it/∼baruffa/WiSHFUL/

These 9 TS files have been generated with a payload bit rate and transmission parameters configuration that can be extracted directly from the file names. In case that none of the files above satisfies the selected payload bit rate, then it is possible to generate a corresponding TS using the OpenCaster and FFmpeg based procedure. To this purpose, example scripts that can be used to compress and multiplex the input clips are included in the "script/dvbt/-TS" folder: there is a batch file (ts_example.bat) that can be used to generate the program and transport streams using the selected compression methods, as well as a python file (ts_example.py) that cam be used to generate the DVB-T TS tables.

1.6 Preparing an Iris XML file

The chosen transmission configuration (input file and transmission parameters) must be specified into an XML document, following the Iris framework XML syntax. It is suggested to copy and modify one of the configuration files that are included in the "examples/dvbt" folder. For instance, the file named "demo_typical_8K.xml" already contains the configuration parameters necessary to perform the transmission of the typical TS. The syntax of the XML file is pretty self-explanatory, and the values associated to parameters such as OFDM mode, cyclic prefix, code rate, etc., can be modified to suit one's needs: during this step, please be advised that the same parameters could have to be modified inside of several components/engines. The following configuration is excerpted from the XML file used to configure DVB-TX-IRIS for performing the transmission of the typical TS, "demo_typical_8K.xml".

```
<?xml version="1.0" encoding="utf-8" ?>

<softwareradio name="Radio1">

 <engine name="phyengine1" class="phyengine">

   <component name="filerawreader1" class="filerawreader">
     <parameter name="filename" value="hd3typ.ts">
     <parameter name="blocksize" value="18800">
     <parameter name="datatype" value="uint8_t">
     <port name="output1" class="output">
   </component>

 </engine>
 <engine name="phyengine2" class="phyengine">

   <component name="dvbt1scrambler1" class="dvbt1scrambler">
     <parameter name="debug" value="false">
     <parameter name="reportinterval" value="5">
     <port name="input1" class="input">
     <port name="output1" class="output">
   </component>

 </engine>
```

```
<engine name="phyengine3" class="phyengine">

  <component name="dvbt1rsencoder1" class="dvbt1rsencoder">
    <parameter name="debug" value="false">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine4" class="phyengine">

  <component name="dvbt1convinterleaver1" class="dvbt1convinterleaver">
    <parameter name="debug" value="false">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine5" class="phyengine">

  <component name="dvbt1convencoder1" class="dvbt1convencoder">
    <parameter name="debug" value="false">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine6" class="phyengine">

 <component name="dvbt1puncturer1" class="dvbt1puncturer">
    <parameter name="debug" value="false">
    <parameter name="coderate" value="34">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine7" class="phyengine">

  <component name="dvbt1bitinterleaver1" class="dvbt1bitinterleaver">
    <parameter name="debug" value="false">
    <parameter name="qammapping" value="64">
    <parameter name="hyerarchymode" value="0">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine8" class="phyengine">

  <component name="dvbt1symbolinterleaver1" class="dvbt1symbolinterleaver">
    <parameter name="debug" value="false">
    <parameter name="ofdmmode" value="8192">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine9" class="phyengine">

  <component name="dvbt1mapper1" class="dvbt1mapper">
    <parameter name="debug" value="false">
    <parameter name="qammapping" value="64">
    <parameter name="hyerarchymode" value="0">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine10" class="phyengine">

 <component name="dvbt1framer1" class="dvbt1framer">
    <parameter name="debug" value="false">
    <parameter name="ofdmmode" value="8192">
    <parameter name="qammapping" value="64">
    <parameter name="hyerarchymode" value="0">
    <parameter name="cellid" value="-1">
    <parameter name="hpcoderate" value="34">
    <parameter name="indepthinterleaver" value="false">
    <parameter name="deltamode" value="4">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine11" class="phyengine">

  <component name="dvbt1ofdmmod1" class="dvbt1ofdmmod">
    <parameter name="debug" value="false">
    <parameter name="ofdmmode" value="8192">
    <parameter name="deltamode" value="4">
    <parameter name="outpower" value="30">
    <parameter name="dacsamplerate" value="12500000">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine12" class="phyengine">

  <component name="dvbt1interpolator1" class="dvbt1interpolator">
    <parameter name="debug" value="false">
    <parameter name="insamplerate" value="0">
    <parameter name="outsamplerate" value="12500000">
    <parameter name="responsefile" value="interp.txt">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine13" class="phyengine">

  <component name="dvbt1filter1" class="dvbt1filter">
    <parameter name="debug" value="false">
    <parameter name="samplerate" value="12500000">
    <parameter name="attenuation" value="25">
    <parameter name="stopband" value="4500000">
    <port name="input1" class="input">
    <port name="output1" class="output">
  </component>

</engine>
<engine name="phyengine14" class="phyengine">

  <component name="usrptx1" class="dvbt1usrptx">
    <parameter name="frequency" value="666000000">
```

```
      <parameter name="fixlooffset" value="5000000">
      <parameter name="rate" value="12500000">
      <parameter name="streaming" value="false">
      <parameter name="gain" value="10">
      <parameter name="numbuffers" value="4">
      <parameter name="args" value="addr=192.168.10.3">
      <port name="input1" class="input">
    </component>

  </engine>

  <link source="filerawreader1.output1" sink="dvbt1scrambler1.input1">
  <link source="dvbt1scrambler1.output1" sink="dvbt1rsencoder1.input1">
  <link source="dvbt1rsencoder1.output1" sink="dvbt1convinterleaver1.input1">
  <link source="dvbt1convinterleaver1.output1" sink="dvbt1convencoder1.input1">
  <link source="dvbt1convencoder1.output1" sink="dvbt1puncturer1.input1">
  <link source="dvbt1puncturer1.output1" sink="dvbt1bitinterleaver1.input1">
  <link source="dvbt1bitinterleaver1.output1" sink="dvbt1symbolinterleaver1.input1">
  <link source="dvbt1symbolinterleaver1.output1" sink="dvbt1mapper1.input1">
  <link source="dvbt1mapper1.output1" sink="dvbt1framer1.input1">
  <link source="dvbt1framer1.output1" sink="dvbt1ofdmmod1.input1">
  <link source="dvbt1ofdmmod1.output1" sink="dvbt1interpolator1.input1">
  <link source="dvbt1interpolator1.output1" sink="dvbt1filter1.input1">
  <link source="dvbt1filter1.output1" sink="usrptx1.input1">

</softwareradio>
```

1.7 USRP setup

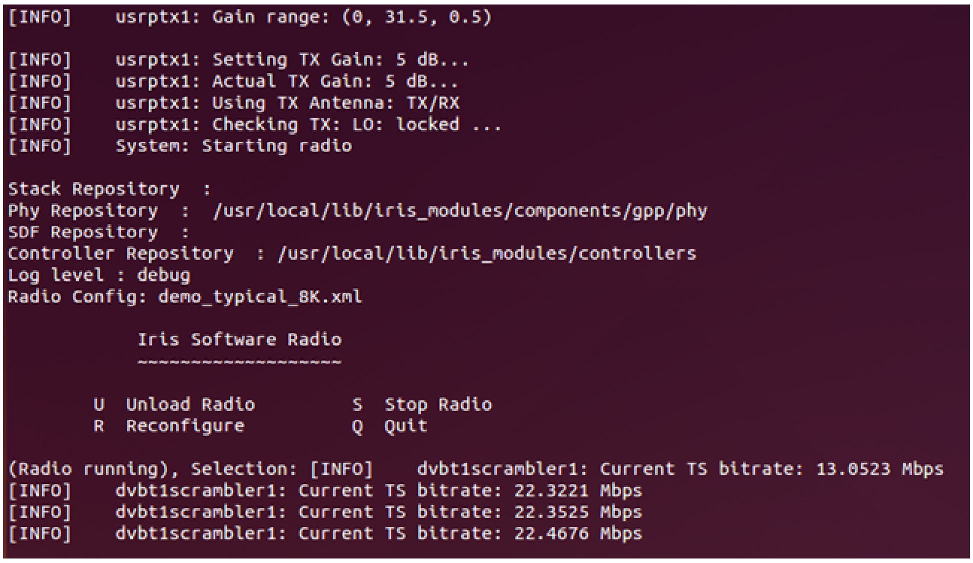
The USRP device must be connected via Gigabit Ethernet to the host PC where Iris and DVB-TX-IRIS are installed. The address of the used USRP device, once it is verified to be reachable via ping commands, can be written in the configuration file. The compatibility between the device firmware revision and the Ettus UHD drivers used to communicate with the USRP should be verified; differently, the device will refuse to work. Either a directional or an omni-directional antenna can be used to transmit the signal, after taking proper care of the USRP device gain value in the XML file.

1.8 Transmit

At this point the RF signal broadcasting can be initiated. The command line to give is

```
iris -f config.iris demo_typical_8K.xml
```

or different if the modified file name is different. If the transmission is proceeding correctly, the command line output should be devoid of USRP communication errors ("U" characters are printed in case of buffer under-run, typically happening when BB digital samples are not being fed sufficiently fast to the USRP) and the SDR modulator should print, periodically, a report of the actually processed bit rate: this value should match the theoretical payload bit rate.



```
[INFO]    usrptx1: Gain range: (0, 31.5, 0.5)

[INFO]    usrptx1: Setting TX Gain: 5 dB...
[INFO]    usrptx1: Actual TX Gain: 5 dB...
[INFO]    usrptx1: Using TX Antenna: TX/RX
[INFO]    usrptx1: Checking TX: LO: locked ...
[INFO]    System: Starting radio

Stack Repository  :
Phy Repository  :  /usr/local/lib/iris_modules/components/gpp/phy
SDF Repository  :
Controller Repository  : /usr/local/lib/iris_modules/controllers
Log level : debug
Radio Config: demo_typical_8K.xml

             Iris Software Radio
             ~~~~~~~~~~~~~~~~~~~

        U  Unload Radio        S  Stop Radio
        R  Reconfigure         Q  Quit

(Radio running), Selection: [INFO]    dvbt1scrambler1: Current TS bitrate: 13.0523 Mbps
[INFO]    dvbt1scrambler1: Current TS bitrate: 22.3221 Mbps
[INFO]    dvbt1scrambler1: Current TS bitrate: 22.3525 Mbps
[INFO]    dvbt1scrambler1: Current TS bitrate: 22.4676 Mbps
```

Figure 1.2: Output of the Iris command line during correct operation.

1.9 Receive and validate

Any standard compliant DVB-T receiver that is in the range covered by the transmitting hardware should be able to pick-up and decode the signal. To this purpose, a full channel scan should be performed in the receiver's setup menu, and one or more TV channels from the WiSHFUL transmission network should be now present in the receiver channel list. The same thing can also be carried out using a DVB-T USB dongle: by this means, it can be possible also to analyze the quality and validate the received signal by a number of parameters such as the signal power, the residual bit error rate, the number of uncorrected packets, etc.

Power loading can also be tested: the "examples/dvbt" folder already contains a demonstration XML configuration file, demo_typical_8K_PL_USRP.xml, as well as a pre-computed power profile file, logo_profile.txt. Additionally, each user can recreate a logo-resembling power profile by running the MATLB/Octave script powerload_logo.m, which is saved in the "scripts/dvbt/MATLAB" folder

1.10 Off-line validation

This step is not generally required, since if the TV signal is correctly received on a TV set, this should be more than sufficient. Anyway, during the compilation step, it is possible to carry out a validation of the DVB-TX-IRIS module C++ components against a MATLAB/Octave implementation of the same component. For every component, there is an associated M-script that can be used to generate random input and output test vectors for the specified component (the M-script and already generated test vectors are present in the "test" folder inside the main component folder). During the build process, the ctest command triggered during the invocation of "make test" executes an automated check of the components correct operation: the input test vector is loaded by the component and transformed into the respective output vector (as per the processing performed by the block); then, the MATLAB output test vector and the Iris output test vector are compared. The test passes or fails depending on the correspondence between these two test vectors.

1.11 Bibliography

1. G. Baruffa, L. Rugini, and P. Banelli, *Design and validation of a Software Defined Radio testbed for DVB-T transmission*, Radioengineering, vol. 23, no. 1, pp. 387–398, Apr. 2014.

2. DVB PROJECT. *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, ETSI EN 300 744 V1.6.1 (2009-01). 2009. Available at: http://www.dvb.org.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 iris Namespace Reference

Namespaces

- phy

6.2 iris::phy Namespace Reference

Classes

- class Dvbt1BitInterleaverComponent

  A DVB-T1 bit interleaver component.
- class Dvbt1ConvEncoderComponent

  A DVB-T1 convolutional encoder component.
- class Dvbt1ConvInterleaverComponent

  A DVB-T1 convolutional interleaver component.
- class Dvbt1FilterComponent

  A DVB-T1 filter component.
- class Dvbt1FormatterComponent

  A DVB-T1 formatter component.
- class Dvbt1FramerComponent

  A DVB-T1 framer component.
- class Dvbt1InterpolatorComponent

  A DVB-T1 interpolator component.
- class Dvbt1MapperComponent

  A DVB-T1 mapper component.
- class Dvbt1NoiseGeneratorComponent

  A DVB-T1 noise generator.
- class Dvbt1OfdmModComponent

  A DVB-T1 OFDM modulator component.
- class Dvbt1PuncturerComponent

  A DVB-T1 puncturer component.
- class Dvbt1RSEncoderComponent

  A DVB-T1 R-S Encoder component.
- class Dvbt1ScramblerComponent

A DVB-T energy dispersal component.

- class Dvbt1SymbolInterleaverComponent

  A DVB-T1 symbol interleaver component.

- class Dvbt1UsrpTxComponent

  The Dvbt1UsrpTx component.

Functions

- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1BitInterleaverComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ConvEncoderComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ConvInterleaverComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FilterComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FormatterComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FramerComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1InterpolatorComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1MapperComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1NoiseGeneratorComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1OfdmModComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1PuncturerComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1RSEncoderComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ScramblerComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1SymbolInterleaverComponent)
- IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1UsrpTxComponent)

6.2.1 Function Documentation

6.2.1.1 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1UsrpTxComponent)

6.2.1.2 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1InterpolatorComponent)

6.2.1.3 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1SymbolInterleaverComponent)

6.2.1.4 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1ConvEncoderComponent)

6.2.1.5 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1RSEncoderComponent)

6.2.1.6 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1FilterComponent)

6.2.1.7 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1MapperComponent)

6.2.1.8 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1NoiseGeneratorComponent)

6.2.1.9 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1PuncturerComponent)

6.2.1.10 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1ConvInterleaverComponent)

6.2.1.11 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1FramerComponent)

6.2.1.12 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1BitInterleaverComponent)

6.2.1.13 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1ScramblerComponent)

6.2.1.14 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1FormatterComponent)

6.2.1.15 iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent , Dvbt1OfdmModComponent)

Chapter 7

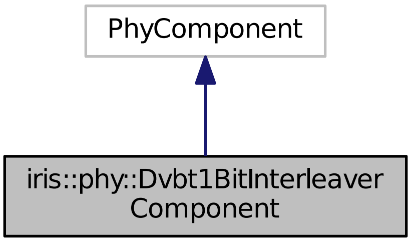
Class Documentation

7.1 iris::phy::Dvbt1BitInterleaverComponent Class Reference

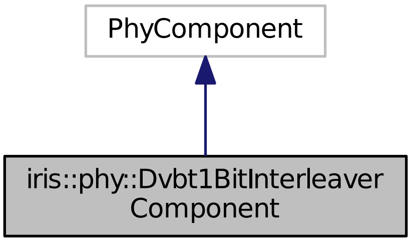
A DVB-T1 bit interleaver component.

`#include <Dvbt1BitInterleaverComponent.h>`

Inheritance diagram for iris::phy::Dvbt1BitInterleaverComponent:



Collaboration diagram for iris::phy::Dvbt1BitInterleaverComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1BitInterleaverComponent (std::string name)

  Default constructor.
- ∼Dvbt1BitInterleaverComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the interleaver ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all offsets, clean registers.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int qamMapping_x

  QAM constellation mapping (default = 16)
- int hyerarchyMode_x

  Hyerarchical mode (default = 0)
- double timeStamp_

  Timestamp of current frame.

- double sampleRate_

  Sample rate of current frame.
- int intOffset_ [2]

  Interleaving offsets (HP & LP)
- int intLength_ [2]

  Interleaving registers length (HP & LP)
- uint8_t ∗ intRegister_ [2]

  Interleaving registers (HP & LP)
- int nu_

  Bits per modulated symbol.

Static Private Attributes

- static int address_v2 [252]

  The interleaving addresses for QPSK.
- static int address_v4 [504]

  The interleaving addresses for 16-QAM.
- static int address_v6 [756]

  The interleaving addresses for 64-QAM.

7.1.1 Detailed Description

A DVB-T1 bit interleaver component.

Dvbt1BitInterleaverComponent is the sixth block composing the DVB-T transmission chain. Its purpose, together with the symbol interleaver, is that of reordering the channel encoded bits in order to convert the possible error bursts arising from the communication on the physical channel (due to impulsive noise, multipath, fading) into well-separated single-error events. This way, the channel decoders at the RX side (Viterbi and Reed-Solomon decoder) are able to perform at their best theoretical limit in white Gaussian noise (WGN) conditions.

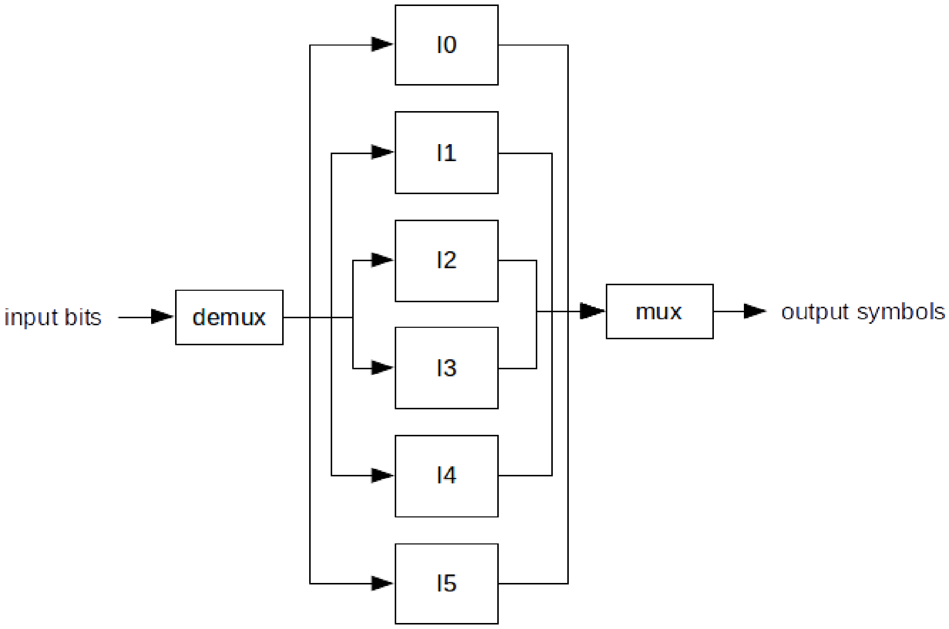


Figure 7.1: DVB-T bit interleaver.

With reference to the figure above, the input demultiplexer routes the incoming bits emitted by the puncturer towards one of the 6 bit interleaving RAMs. Every RAM has a capacity of 126 bits. When all the RAMs are filled, the stored bits are read out according to a particular cyclic address shift and composed into v-bit symbols, where v is the number of bits of the particular M-QAM mapping adopted. Please note that only v RAM interleavers are adopted, thus the figure above refers to the 64-QAM case. This block accepts in input elements in uint8_t (bits) and generates in output v-bit symbols (uint8_t).

There are three parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *qammapping*: by default set to "16", this is used to select one of the three possible QAM mappings. The admitted values are "4", "16", "64".
- *hyerarchymode*: by default set to "0", which means "not hyerarchical". Hierarchical modes are used to transmit two different transport streams, one with a high priority (HP) information and another one with a low priority (LP) information. The admitted values are "0, "1", "2", "4". NOTE: hyerarchical modes are not implemented in the current release of this modulator.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at ETSI Publications Download Area

Definition at line 88 of file Dvbt1BitInterleaverComponent.h.

7.1.2 Member Typedef Documentation

7.1.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1BitInterleaverComponent::ByteVec

A vector of bytes.

Definition at line 94 of file Dvbt1BitInterleaverComponent.h.

7.1.2.2 typedef ByteVec::iterator iris::phy::Dvbt1BitInterleaverComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 97 of file Dvbt1BitInterleaverComponent.h.

7.1.3 Constructor & Destructor Documentation

7.1.3.1 iris::phy::Dvbt1BitInterleaverComponent::Dvbt1BitInterleaverComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1BitInterleaverComponent.cpp.

References begin(), debug_x, end(), hyerarchyMode_x, intRegister_, and qamMapping_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1bitinterleaver",                 // component type
                "A DVB-T1 bit interleaver component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  int qamarr[] = {4,16,64};
71   registerParameter(
72     "qammapping", "QAM constellation mapping",
73     "16", true, qamMapping_x, list<int>(begin(qamarr),end(qamarr)));
74
75   int harr[] = {0,1,2,4};
76   registerParameter(
77     "hyerarchymode", "Hyerarchical mode (0 = NH)",
78     "0", true, hyerarchyMode_x, list<int>(begin(harr),end(harr)));
79
80   intRegister_[0] = NULL;
81   intRegister_[1] = NULL;
82 }
```

7.1.3.2 iris::phy::Dvbt1BitInterleaverComponent::~Dvbt1BitInterleaverComponent ( )

Default destructor.

Just calls destroy().

Definition at line 87 of file Dvbt1BitInterleaverComponent.cpp.

References destroy().

```
88 {
89   destroy();
90 }
```

7.1.4 Member Function Documentation

7.1.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1BitInterleaverComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 133 of file Dvbt1BitInterleaverComponent.h.

Referenced by Dvbt1BitInterleaverComponent().

```
133 { return &arr[0]; }
```

7.1.4.2 void iris::phy::Dvbt1BitInterleaverComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 106 of file Dvbt1BitInterleaverComponent.cpp.

```
109 {
110   outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
111 }
```

7.1.4.3 void iris::phy::Dvbt1BitInterleaverComponent::destroy ( ) [private]

Destroy the component.

Definition at line 320 of file Dvbt1BitInterleaverComponent.cpp.

References intRegister_.

Referenced by parameterHasChanged(), and ~Dvbt1BitInterleaverComponent().

```
321 {
322   // clean
323   delete[] intRegister_[0];
324   delete[] intRegister_[1];
325 }
```

7.1.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1BitInterleaverComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 135 of file Dvbt1BitInterleaverComponent.h.

Referenced by Dvbt1BitInterleaverComponent().

```
135 { return &arr[0]+N; }
```

7.1.4.5 void iris::phy::Dvbt1BitInterleaverComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 116 of file Dvbt1BitInterleaverComponent.cpp.

References setup().

```
117 {
118   setup();
119 }
```

7.1.4.6 void iris::phy::Dvbt1BitInterleaverComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has two significant parameters

Definition at line 290 of file Dvbt1BitInterleaverComponent.cpp.

References destroy(), and setup().

```
291 {
292   if(name == "qammapping" || name == "hyerarchymode")
293   {
294     destroy();
295     setup();
296   }
297 }
```

7.1.4.7 void iris::phy::Dvbt1BitInterleaverComponent::process ( ) [virtual]

Main processing method.

Definition at line 204 of file Dvbt1BitInterleaverComponent.cpp.

References address_v2, address_v4, address_v6, debug_x, hyerarchyMode_x, intLength_, intOffset_, intRegister_, nu_, and qamMapping_x.

```
205 {
206   // request input
207   DataSet< uint8_t > *in1 = NULL;
208   getInputDataSet("input1", in1);
209
210   // calculate sizes
211   int in1size = in1 ? (int) in1->data.size() : 0;
```

```
  int outsize = intLength_[0] * ((in1size + intOffset_[0]) /
  intLength_[0]) / nu_;

  // request output
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in1/out: " << in1size << "/" << outsize;

  // bit by bit
  for(ByteVecIt in1it = in1->data.begin(), outit = out->data.begin();
    in1it < in1->data.end(); in1it++)
  {
    // copy to register
    intRegister_[0][intOffset_[0]++] = *in1it;

    // trigger interleaving
    if(intOffset_[0] == intLength_[0])
    {
      // reset offset
      intOffset_[0] = 0;

      // do the copy
      if(hyerarchyMode_x == 0)
      {
        switch(qamMapping_x)
        {
          // read back according to the QAM mode and compose the output symbol
          case 4:
            for(int b = 0; b < intLength_[0]; b += 2)
            {
              *outit++ =
                (intRegister_[0][address_v2[b + 0]] << 1) |
                (intRegister_[0][address_v2[b + 1]] << 0);
            }
            break;
          case 16:
            for(int b = 0; b < intLength_[0]; b += 4)
            {
              *outit++ =
                (intRegister_[0][address_v4[b + 0]] << 3) |
                (intRegister_[0][address_v4[b + 1]] << 2) |
                (intRegister_[0][address_v4[b + 2]] << 1) |
                (intRegister_[0][address_v4[b + 3]] << 0);
            }
            break;
          case 64:
            for(int b = 0; b < intLength_[0]; b += 6)
            {
              *outit++ =
                (intRegister_[0][address_v6[b + 0]] << 5) |
                (intRegister_[0][address_v6[b + 1]] << 4) |
                (intRegister_[0][address_v6[b + 2]] << 3) |
                (intRegister_[0][address_v6[b + 3]] << 2) |
                (intRegister_[0][address_v6[b + 4]] << 1) |
                (intRegister_[0][address_v6[b + 5]] << 0);
            }
            break;
          default:
            LOG(LERROR) << "Unsupported QAM mapping";
        }
      }
    }
  }

  // copy the timestamp and sample rate for the DataSets
  out->timeStamp = in1->timeStamp;
  out->sampleRate = in1->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in1);
  releaseOutputDataSet("output1", out);
}
```

7.1.4.8 void iris::phy::Dvbt1BitInterleaverComponent::registerPorts ( ) [virtual]

Register the interleaver ports with the IRIS system.

This component has two inputs that accept bits (one bit per byte) and one output that provides symbols (in one byte).

Definition at line 96 of file Dvbt1BitInterleaverComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerInputPort("input2", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.1.4.9 void iris::phy::Dvbt1BitInterleaverComponent::setup ( ) [private]

Set up all offsets, clean registers.

Definition at line 300 of file Dvbt1BitInterleaverComponent.cpp.

References hyerarchyMode_x, intLength_, intOffset_, intRegister_, nu_, and qamMapping_x.

Referenced by initialize(), and parameterHasChanged().

```
{
  // clean
  intOffset_[0] = 0;
  intOffset_[1] = 0;

  // modulation order
  nu_ = qamMapping_x == 4 ? 2 : (qamMapping_x == 16 ? 4 : 6);

  // lengths
  intLength_[0] = (hyerarchyMode_x == 0 ? (126 * nu_) : 126 * 2);
  intLength_[1] = (hyerarchyMode_x == 0 ? 1 /* to avoid divide by 0 error */
    : (126 * (nu_ - 2)));

  // alloc
  intRegister_[0] = new uint8_t [intLength_[0]];
  intRegister_[1] = new uint8_t [intLength_[1]];
}
```

7.1.5 Member Data Documentation

7.1.5.1 int iris::phy::Dvbt1BitInterleaverComponent::address_v2 [static], [private]

Initial value:

```
= {
    0, 127,   2, 129,   4, 131,   6, 133,   8, 135,  10, 137,  12, 139,  14, 141,  16, 143,  18, 145,  20,
     147,
   22, 149,  24, 151,  26, 153,  28, 155,  30, 157,  32, 159,  34, 161,  36, 163,  38, 165,  40, 167,  42,
      169,
   44, 171,  46, 173,  48, 175,  50, 177,  52, 179,  54, 181,  56, 183,  58, 185,  60, 187,  62, 189,  64,
      191,
   66, 193,  68, 195,  70, 197,  72, 199,  74, 201,  76, 203,  78, 205,  80, 207,  82, 209,  84, 211,  86,
      213,
   88, 215,  90, 217,  92, 219,  94, 221,  96, 223,  98, 225, 100, 227, 102, 229, 104, 231, 106, 233, 108,
      235,
  110, 237, 112, 239, 114, 241, 116, 243, 118, 245, 120, 247, 122, 249, 124, 251, 126,   1, 128,   3, 130
    ,   5,
  132,   7, 134,   9, 136,  11, 138,  13, 140,  15, 142,  17, 144,  19, 146,  21, 148,  23, 150,  25, 152
    ,  27,
  154,  29, 156,  31, 158,  33, 160,  35, 162,  37, 164,  39, 166,  41, 168,  43, 170,  45, 172,  47, 174
    ,  49,
  176,  51, 178,  53, 180,  55, 182,  57, 184,  59, 186,  61, 188,  63, 190,  65, 192,  67, 194,  69, 196
    ,  71,
  198,  73, 200,  75, 202,  77, 204,  79, 206,  81, 208,  83, 210,  85, 212,  87, 214,  89, 216,  91, 218
    ,  93,
  220,  95, 222,  97, 224,  99, 226, 101, 228, 103, 230, 105, 232, 107, 234, 109, 236, 111, 238, 113, 240
    , 115,
  242, 117, 244, 119, 246, 121, 248, 123, 250, 125
}
```

The interleaving addresses for QPSK.

Definition at line 127 of file Dvbt1BitInterleaverComponent.h.

Referenced by process().

7.1.5.2 int iris::phy::Dvbt1BitInterleaverComponent::address_v4 [static], [private]

Initial value:

```
= {
    0, 254, 421, 171,   4, 258, 425, 175,   8, 262, 429, 179,  12, 266, 433, 183,  16, 270, 437, 187,  20,
      274,
    441, 191,  24, 278, 445, 195,  28, 282, 449, 199,  32, 286, 453, 203,  36, 290, 457, 207,  40, 294, 461
      , 211,
    44, 298, 465, 215,  48, 302, 469, 219,  52, 306, 473, 223,  56, 310, 477, 227,  60, 314, 481, 231,  64,
       318,
    485, 235,  68, 322, 489, 239,  72, 326, 493, 243,  76, 330, 497, 247,  80, 334, 501, 251,  84, 338,   1
      , 255,
    88, 342,   5, 259,  92, 346,   9, 263,  96, 350,  13, 267, 100, 354,  17, 271, 104, 358,  21, 275, 108,
       362,
    25, 279, 112, 366,  29, 283, 116, 370,  33, 287, 120, 374,  37, 291, 124, 378,  41, 295, 128, 382,  45,
       299,
    132, 386,  49, 303, 136, 390,  53, 307, 140, 394,  57, 311, 144, 398,  61, 315, 148, 402,  65, 319, 152
      , 406,
    69, 323, 156, 410,  73, 327, 160, 414,  77, 331, 164, 418,  81, 335, 168, 422,  85, 339, 172, 426,  89,
       343,
    176, 430,  93, 347, 180, 434,  97, 351, 184, 438, 101, 355, 188, 442, 105, 359, 192, 446, 109, 363, 196
      , 450,
    113, 367, 200, 454, 117, 371, 204, 458, 121, 375, 208, 462, 125, 379, 212, 466, 129, 383, 216, 470, 133
      , 387,
    220, 474, 137, 391, 224, 478, 141, 395, 228, 482, 145, 399, 232, 486, 149, 403, 236, 490, 153, 407, 240
      , 494,
    157, 411, 244, 498, 161, 415, 248, 502, 165, 419, 252,   2, 169, 423, 256,   6, 173, 427, 260,  10, 177
      , 431,
    264,  14, 181, 435, 268,  18, 185, 439, 272,  22, 189, 443, 276,  26, 193, 447, 280,  30, 197, 451, 284
      ,  34,
    201, 455, 288,  38, 205, 459, 292,  42, 209, 463, 296,  46, 213, 467, 300,  50, 217, 471, 304,  54, 221
      , 475,
    308,  58, 225, 479, 312,  62, 229, 483, 316,  66, 233, 487, 320,  70, 237, 491, 324,  74, 241, 495, 328
      ,  78,
    245, 499, 332,  82, 249, 503, 336,  86, 253,   3, 340,  90, 257,   7, 344,  94, 261,  11, 348,  98, 265
      ,  15,
    352, 102, 269,  19, 356, 106, 273,  23, 360, 110, 277,  27, 364, 114, 281,  31, 368, 118, 285,  35, 372
      , 122,
    289,  39, 376, 126, 293,  43, 380, 130, 297,  47, 384, 134, 301,  51, 388, 138, 305,  55, 392, 142, 309
      ,  59,
    396, 146, 313,  63, 400, 150, 317,  67, 404, 154, 321,  71, 408, 158, 325,  75, 412, 162, 329,  79, 416
      , 166,
    333,  83, 420, 170, 337,  87, 424, 174, 341,  91, 428, 178, 345,  95, 432, 182, 349,  99, 436, 186, 353
      , 103,
    440, 190, 357, 107, 444, 194, 361, 111, 448, 198, 365, 115, 452, 202, 369, 119, 456, 206, 373, 123, 460
      , 210,
    377, 127, 464, 214, 381, 131, 468, 218, 385, 135, 472, 222, 389, 139, 476, 226, 393, 143, 480, 230, 397
      , 147,
    484, 234, 401, 151, 488, 238, 405, 155, 492, 242, 409, 159, 496, 246, 413, 163, 500, 250, 417, 167
}
```

The interleaving addresses for 16-QAM.

Definition at line 128 of file Dvbt1BitInterleaverComponent.h.

Referenced by process().

7.1.5.3 int iris::phy::Dvbt1BitInterleaverComponent::address_v6 [static], [private]

The interleaving addresses for 64-QAM.

Definition at line 129 of file Dvbt1BitInterleaverComponent.h.

Referenced by process().

7.1.5.4 bool iris::phy::Dvbt1BitInterleaverComponent::debug_x [private]

Debug flag (default = false)

Definition at line 111 of file Dvbt1BitInterleaverComponent.h.

Referenced by Dvbt1BitInterleaverComponent(), and process().

7.1.5.5 int iris::phy::Dvbt1BitInterleaverComponent::hyerarchyMode_x [private]

Hyerarchical mode (default = 0)

Definition at line 113 of file Dvbt1BitInterleaverComponent.h.

Referenced by Dvbt1BitInterleaverComponent(), process(), and setup().

7.1.5.6 int iris::phy::Dvbt1BitInterleaverComponent::intLength_[2] [private]

Interleaving registers length (HP & LP)

Definition at line 122 of file Dvbt1BitInterleaverComponent.h.

Referenced by process(), and setup().

7.1.5.7 int iris::phy::Dvbt1BitInterleaverComponent::intOffset_[2] [private]

Interleaving offsets (HP & LP)

Definition at line 121 of file Dvbt1BitInterleaverComponent.h.

Referenced by process(), and setup().

7.1.5.8 uint8_t∗ iris::phy::Dvbt1BitInterleaverComponent::intRegister_[2] [private]

Interleaving registers (HP & LP)

Definition at line 123 of file Dvbt1BitInterleaverComponent.h.

Referenced by destroy(), Dvbt1BitInterleaverComponent(), process(), and setup().

7.1.5.9 int iris::phy::Dvbt1BitInterleaverComponent::nu_ [private]

Bits per modulated symbol.

Definition at line 124 of file Dvbt1BitInterleaverComponent.h.

Referenced by process(), and setup().

7.1.5.10 int iris::phy::Dvbt1BitInterleaverComponent::qamMapping_x [private]

QAM constellation mapping (default = 16)

Definition at line 112 of file Dvbt1BitInterleaverComponent.h.

Referenced by Dvbt1BitInterleaverComponent(), process(), and setup().

7.1.5.11 double iris::phy::Dvbt1BitInterleaverComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 119 of file Dvbt1BitInterleaverComponent.h.

7.1.5.12 double iris::phy::Dvbt1BitInterleaverComponent::timeStamp_ [private]

Timestamp of current frame.

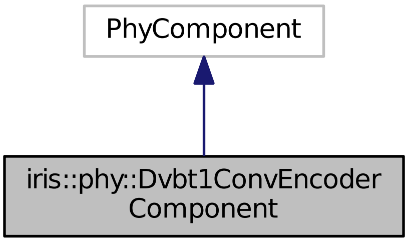
Definition at line 118 of file Dvbt1BitInterleaverComponent.h.

7.2 iris::phy::Dvbt1ConvEncoderComponent Class Reference

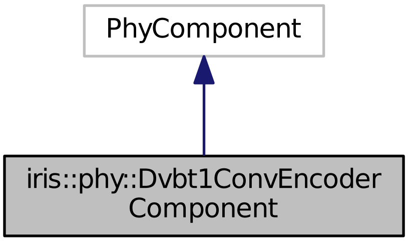
A DVB-T1 convolutional encoder component.

`#include <Dvbt1ConvEncoderComponent.h>`

Inheritance diagram for iris::phy::Dvbt1ConvEncoderComponent:



Collaboration diagram for iris::phy::Dvbt1ConvEncoderComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1ConvEncoderComponent (std::string name)

  Default constructor.
- ∼Dvbt1ConvEncoderComponent ()

  Default destructor.

- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the scrambler ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Clean variables.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T $*$ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T $*$ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int status_

  Register with the delayed inputs (state)

Static Private Attributes

- static unsigned char parity_ [256]

  LUT containing the parity bits.

7.2.1 Detailed Description

A DVB-T1 convolutional encoder component.

Dvbt1ConvEncoderComponent is the fourth block composing the DVB-T transmission chain. This block is a binary convolutional encoder of rate $k/n = 1/2$ with a constraint length of $L = 6$. The output bits are generated by proper connections between the shift register cells and the XOR-adders. The connection configuration is represented in octal form by the generator polynomials $G_1 = 0171$ and $G_2 = 0133$.

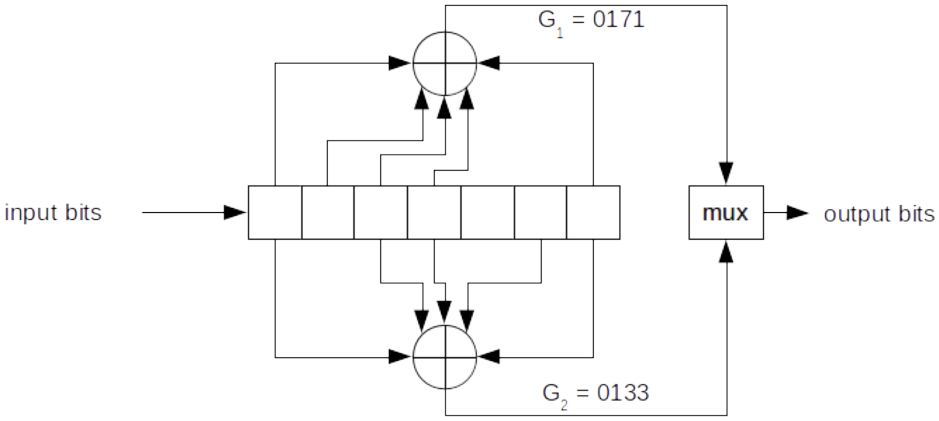


Figure 7.2: DVB-T convolutional encoder.

This block accepts in input elements in uint8_t (octets of bits) and generates in output single bits (always formatted as uint8_t). There is only one parameter that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at ETSI Publications Download Area
- S. Li, D. J. Costello, *Error Control Coding, Second Edition*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2004

Definition at line 72 of file Dvbt1ConvEncoderComponent.h.

7.2.2 Member Typedef Documentation

7.2.2.1 typedef std::vector<uint8_t> **iris::phy::Dvbt1ConvEncoderComponent::ByteVec**

A vector of bytes.

Definition at line 78 of file Dvbt1ConvEncoderComponent.h.

7.2.2.2 typedef ByteVec::iterator **iris::phy::Dvbt1ConvEncoderComponent::ByteVecIt**

An iterator for a vector of bytes.

Definition at line 81 of file Dvbt1ConvEncoderComponent.h.

7.2.3 Constructor & Destructor Documentation

7.2.3.1 iris::phy::Dvbt1ConvEncoderComponent::Dvbt1ConvEncoderComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1ConvEncoderComponent.cpp.

References debug_x.

```
  : PhyComponent(name,                        // component name
                 "dvbt1convencoder",            // component type
                 "A DVB-T1 convolutional encoder component", // description
                 "Giuseppe Baruffa",            // author
                 "0.1")                       // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,status_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);
}
```

7.2.3.2 iris::phy::Dvbt1ConvEncoderComponent::∼Dvbt1ConvEncoderComponent ( )

Default destructor.

Just calls destroy().

Definition at line 75 of file Dvbt1ConvEncoderComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.2.4 Member Function Documentation

7.2.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ConvEncoderComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 109 of file Dvbt1ConvEncoderComponent.h.

```
{ return &arr[0]; }
```

7.2.4.2 void iris::phy::Dvbt1ConvEncoderComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 93 of file Dvbt1ConvEncoderComponent.cpp.

```
{
  outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
}
```

7.2.4.3 void iris::phy::Dvbt1ConvEncoderComponent::destroy ( ) [private]

Destroy the component.

Definition at line 210 of file Dvbt1ConvEncoderComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1ConvEncoderComponent().

```
{
}
```

7.2.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ConvEncoderComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 111 of file Dvbt1ConvEncoderComponent.h.

```
111 { return &arr[0]+N; }
```

7.2.4.5 void iris::phy::Dvbt1ConvEncoderComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 103 of file Dvbt1ConvEncoderComponent.cpp.

References setup().

```
104 {
105   setup();
106 }
```

7.2.4.6 void iris::phy::Dvbt1ConvEncoderComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has no significant parameters

Definition at line 194 of file Dvbt1ConvEncoderComponent.cpp.

References destroy(), and setup().

```
195 {
196   if(name == "???")
197   {
198     destroy();
199     setup();
200   }
201 }
```

7.2.4.7 void iris::phy::Dvbt1ConvEncoderComponent::process ( ) [virtual]

Main processing method.

Definition at line 152 of file Dvbt1ConvEncoderComponent.cpp.

References debug_x, g1, g2, parity_, and status_.

```
153 {
154   // request input
155   DataSet< uint8_t >* in = NULL;
156   getInputDataSet("input1", in);
157
158   // calculate sizes
159   int insize = in ? (int) in->data.size() : 0;
160   int outsize = 2 * 8 * insize;
161
162   // request output - double size
163   DataSet< uint8_t >* out = NULL;
164   getOutputDataSet("output1", out, outsize);
165
166   // print debug info
167   if(debug_x)
168     LOG(LINFO) << "in/out: " << insize << "/" << outsize;
169
170   // iterate over all input bytes
171   for(ByteVecIt init = in->data.begin(), outit = out->data.begin(); init < in->data.end(); init++)
172   {
    // iterate over all the bits of the byte
    for(int j = 7; j >= 0; j--)
    {
      status_ = (status_ << 1) | ((*init >> j) & 0x01); // new status
      *outit++ = parity_[status_ & g1]; // first parity bit
      *outit++ = parity_[status_ & g2]; // second parity bit
    }
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.2.4.8 void iris::phy::Dvbt1ConvEncoderComponent::registerPorts ( ) [virtual]

Register the scrambler ports with the IRIS system.

This component has one input that accepts bytes and one output that provides convolutional encoded bits (one bit per byte).

Definition at line 84 of file Dvbt1ConvEncoderComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.2.4.9 void iris::phy::Dvbt1ConvEncoderComponent::setup ( ) [private]

Clean variables.

Definition at line 204 of file Dvbt1ConvEncoderComponent.cpp.

References status_.

Referenced by initialize(), and parameterHasChanged().

```
{
  status_ = 0;
}
```

7.2.5 Member Data Documentation

7.2.5.1 bool iris::phy::Dvbt1ConvEncoderComponent::debug_x [private]

Debug flag (default = false)

Definition at line 95 of file Dvbt1ConvEncoderComponent.h.

Referenced by Dvbt1ConvEncoderComponent(), and process().

7.2.5.2 unsigned char iris::phy::Dvbt1ConvEncoderComponent::parity_ [static], [private]

LUT containing the parity bits.

This look-up tables contains pairs of convolutional encoder parity bit outputs for all the possible configurations of states (64) and inputs (2)

Definition at line 105 of file Dvbt1ConvEncoderComponent.h.

Referenced by process().

7.2.5.3 double iris::phy::Dvbt1ConvEncoderComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 101 of file Dvbt1ConvEncoderComponent.h.

7.2.5.4 int iris::phy::Dvbt1ConvEncoderComponent::status_ [private]

Register with the delayed inputs (state)

Definition at line 103 of file Dvbt1ConvEncoderComponent.h.

Referenced by process(), and setup().

7.2.5.5 double iris::phy::Dvbt1ConvEncoderComponent::timeStamp_ [private]

Timestamp of current frame.

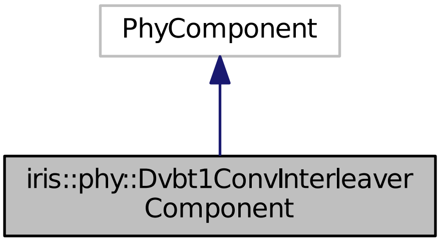
Definition at line 100 of file Dvbt1ConvEncoderComponent.h.

7.3 iris::phy::Dvbt1ConvInterleaverComponent Class Reference

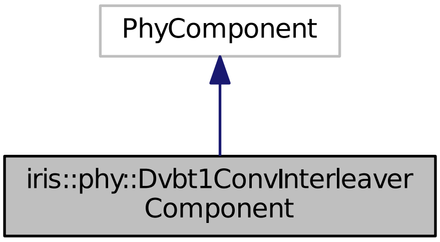
A DVB-T1 convolutional interleaver component.

`#include <Dvbt1ConvInterleaverComponent.h>`

Inheritance diagram for iris::phy::Dvbt1ConvInterleaverComponent:



Collaboration diagram for iris::phy::Dvbt1ConvInterleaverComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1ConvInterleaverComponent (std::string name)

  Default constructor.
- ∼Dvbt1ConvInterleaverComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the interleaver ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up offsets and clean interleaver registers.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int b_ [12]

  Interleaving ststus.
- uint8_t I0_ [1]

  First interleaving register, not used.
- uint8_t I1_ [17]

  Second interleaving register.
- uint8_t I2_ [2 ∗17]

  Third interleaving register.
- uint8_t I3_ [3 ∗17]

  Fourth interleaving register.
- uint8_t I4_ [4 ∗17]

  Fifth interleaving register.
- uint8_t I5_ [5 ∗17]

  Sixth interleaving register.
- uint8_t I6_ [6 ∗17]

  Seventh interleaving register.
- uint8_t I7_ [7 ∗17]

  Eighth interleaving register.
- uint8_t I8_ [8 ∗17]

  Ninth interleaving register.
- uint8_t I9_ [9 ∗17]

  Tenth interleaving register.
- uint8_t I10_ [10 ∗17]

  Eleventh interleaving register.
- uint8_t I11_ [11 ∗17]

  Twelfth interleaving register.
- int rsOffset_

  Input offset.

7.3.1 Detailed Description

A DVB-T1 convolutional interleaver component.

Dvbt1ConvInterleaverComponent is the third block composing the DVB-T transmission chain. The purpose of this interleaver, placed between the R-S encoder and the convolutional encoder, is most useful at decoding time. In fact, the corresponding deinterleaver has the task to shuffle apart consecutive bursts of errors coming out from the Viterbi decoder, so that the R-S error correcting capability (up to 8 bytes in a codeword of 204 bytes) is not exceeded. The convolutional interleaving process is based on the Forney approach, which is compatible with the Ramsey type III approach, with a depth of $I = 12$. Each cell in every interleaving delay path is composed by $M_I = 17$ bytes. The interleaved data bytes are composed of error protected packets and are delimited by inverted or non-inverted MPEG-2 sync bytes (204 bytes periodicity).

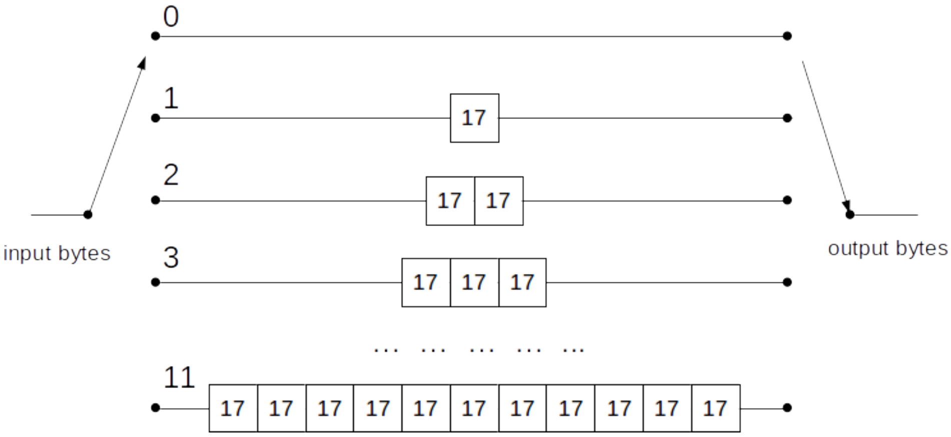


Figure 7.3: DVB-T convolutional interleaver.

Please note that the convolutional interleaver does not operate strictly as a row-column block interleaver, since it keeps memory of older bytes in the current block, and does not emit all the bytes in the current block. For a different implementation of this block operation, please refer also to the testing section implemented in MATLAB. In that case, the operation of this interleaver is performed using a block row-column interleaver that is *slant* after data load and before data dump.

There is only one parameter that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`
- Forney, G. D., *Burst-Correcting Codes for the Classic Bursty Channel*, IEEE Transactions on Communications, vol. COM-19, October 1971, pp. 772-781.
- Ramsey, J. L., *Realization of Optimum Interleavers*, IEEE Transactions on Information Theory, IT-16 (3), May 1970, pp. 338-345.

Definition at line 85 of file Dvbt1ConvInterleaverComponent.h.

7.3.2 Member Typedef Documentation

7.3.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1ConvInterleaverComponent::ByteVec

A vector of bytes.

Definition at line 91 of file Dvbt1ConvInterleaverComponent.h.

7.3.2.2 typedef ByteVec::iterator iris::phy::Dvbt1ConvInterleaverComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 94 of file Dvbt1ConvInterleaverComponent.h.

7.3.3 Constructor & Destructor Documentation

7.3.3.1 iris::phy::Dvbt1ConvInterleaverComponent::Dvbt1ConvInterleaverComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1ConvInterleaverComponent.cpp.

References debug_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1convinterleaver",                 // component type
                "A DVB-T1 convolutional interleaver component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,rsOffset_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);
}
```

7.3.3.2 iris::phy::Dvbt1ConvInterleaverComponent::∼Dvbt1ConvInterleaverComponent ( )

Default destructor.

Just calls destroy().

Definition at line 75 of file Dvbt1ConvInterleaverComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.3.4 Member Function Documentation

7.3.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ConvInterleaverComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 133 of file Dvbt1ConvInterleaverComponent.h.

```
{ return &arr[0]; }
```

7.3.4.2 void iris::phy::Dvbt1ConvInterleaverComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 93 of file Dvbt1ConvInterleaverComponent.cpp.

```
96 {
97   outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
98 }
```

7.3.4.3 void iris::phy::Dvbt1ConvInterleaverComponent::destroy ( ) [private]

Destroy the component.

Definition at line 309 of file Dvbt1ConvInterleaverComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1ConvInterleaverComponent().

```
310 {
311 }
```

7.3.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ConvInterleaverComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 135 of file Dvbt1ConvInterleaverComponent.h.

```
135 { return &arr[0]+N; }
```

7.3.4.5 void iris::phy::Dvbt1ConvInterleaverComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 103 of file Dvbt1ConvInterleaverComponent.cpp.

References setup().

```
104 {
105   setup();
106 }
```

7.3.4.6 void iris::phy::Dvbt1ConvInterleaverComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has no significant parameters

Definition at line 271 of file Dvbt1ConvInterleaverComponent.cpp.

References destroy(), and setup().

```
272 {
273   if(name == "???")
274   {
275     destroy();
276     setup();
277   }
278 }
```

7.3.4.7 void iris::phy::Dvbt1ConvInterleaverComponent::process ( ) [virtual]

Main processing method.

Definition at line 109 of file Dvbt1ConvInterleaverComponent.cpp.

References b_, debug_x, I10_, I11_, I1_, I2_, I3_, I4_, I5_, I6_, I7_, I8_, I9_, and rsOffset_.

```
{
  // request input
  DataSet< uint8_t >* in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = insize;

  // request output
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // process data with a humongous Duff’s device!
  // For more info on what a Duff is, check the Wikipedia page
  // https://en.wikipedia.org/wiki/Duff’s_device
  ByteVecIt init = in->data.begin();
  ByteVecIt outit = out->data.begin();
  switch(rsOffset_)
  {
    // in the following, we load a bunch of bytes in the generic path
    // and advance the relevant pointer
    case 0:
    case 12:
    do
    {
      // first, direct path
      *outit++ = *init++;
      rsOffset_ = 1;
          if(init == in->data.end())
            break;

    case 1:
      // second path
      *outit++ = I1_[b_[1]];
      I1_[b_[1]] = *init++;
      if(++(b_[1]) == 17 * 1)
        b_[1] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 2:
      // third path
      *outit++ = I2_[b_[2]];
      I2_[b_[2]] = *init++;
      if(++(b_[2]) == 17 * 2)
        b_[2] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 3:
      // fourth path
      *outit++ = I3_[b_[3]];
      I3_[b_[3]] = *init++;
      if(++(b_[3]) == 17 * 3)
        b_[3] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 4:
      // fifth path
      *outit++ = I4_[b_[4]];
      I4_[b_[4]] = *init++;
      if(++(b_[4]) == 17 * 4)
        b_[4] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 5:
      // sixth path
      *outit++ = I5_[b_[5]];
      I5_[b_[5]] = *init++;
      if(++(b_[5]) == 17 * 5)
        b_[5] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 6:
          // seventh path
      *outit++ = I6_[b_[6]];
      I6_[b_[6]] = *init++;
      if(++(b_[6]) == 17 * 6)
        b_[6] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 7:
      // eighth path
      *outit++ = I7_[b_[7]];
      I7_[b_[7]] = *init++;
      if(++(b_[7]) == 17 * 7)
        b_[7] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 8:
      // ninth path
      *outit++ = I8_[b_[8]];
      I8_[b_[8]] = *init++;
      if(++(b_[8]) == 17 * 8)
        b_[8] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 9:
      // tenth path
      *outit++ = I9_[b_[9]];
      I9_[b_[9]] = *init++;
      if(++(b_[9]) == 17 * 9)
        b_[9] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 10:
      // eleventh path
      *outit++ = I10_[b_[10]];
      I10_[b_[10]] = *init++;
      if(++(b_[10]) == 17 * 10)
        b_[10] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    case 11:
      // twelfth and final path
      *outit++ = I11_[b_[11]];
      I11_[b_[11]] = *init++;
      if(++(b_[11]) == 17 * 11)
        b_[11] = 0;
      rsOffset_++;
      if(init == in->data.end())
        break;

    } while(true);
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.3.4.8 void iris::phy::Dvbt1ConvInterleaverComponent::registerPorts ( ) [virtual]

Register the interleaver ports with the IRIS system.

This component has one input that accepts bytes and one output that provides interleaved bytes.

Definition at line 84 of file Dvbt1ConvInterleaverComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.3.4.9 void iris::phy::Dvbt1ConvInterleaverComponent::setup ( ) [private]

Set up offsets and clean interleaver registers.

Please note that filling the registers with zeroes, as we do below, generates peaky transients in the final waveform right after the system start-up and up to the moment when all registers are filled by real data bytes. In order to avoid this, we should fill this with a random sequence of bytes, instead of zeroes.

Definition at line 287 of file Dvbt1ConvInterleaverComponent.cpp.

References b_, I0_, I10_, I11_, I1_, I2_, I3_, I4_, I5_, I6_, I7_, I8_, I9_, and rsOffset_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // clean registers
  memset(b_, 0, sizeof(b_));
  memset(I0_, 0, sizeof(I0_));
  memset(I1_, 0, sizeof(I1_));
  memset(I2_, 0, sizeof(I2_));
  memset(I3_, 0, sizeof(I3_));
  memset(I4_, 0, sizeof(I4_));
  memset(I5_, 0, sizeof(I5_));
  memset(I6_, 0, sizeof(I6_));
  memset(I7_, 0, sizeof(I7_));
  memset(I8_, 0, sizeof(I8_));
  memset(I9_, 0, sizeof(I9_));
  memset(I10_, 0, sizeof(I10_));
  memset(I11_, 0, sizeof(I11_));

  // reset the offset
  rsOffset_ = 0;
}
```

7.3.5 Member Data Documentation

7.3.5.1 int iris::phy::Dvbt1ConvInterleaverComponent::b_[12] [private]

Interleaving ststus.

Definition at line 116 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.2 bool iris::phy::Dvbt1ConvInterleaverComponent::debug_x [private]

Debug flag (default = false)

Definition at line 108 of file Dvbt1ConvInterleaverComponent.h.

Referenced by Dvbt1ConvInterleaverComponent(), and process().

7.3.5.3 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I0_[1] [private]

First interleaving register, not used.

Definition at line 117 of file Dvbt1ConvInterleaverComponent.h.

Referenced by setup().

7.3.5.4 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I10_[10 ∗17] [private]

Eleventh interleaving register.

Definition at line 127 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.5 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I11_[11 ∗17] [private]

Twelfth interleaving register.

Definition at line 128 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.6 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I1_[17] [private]

Second interleaving register.

Definition at line 118 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.7 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I2_[2 ∗17] [private]

Third interleaving register.

Definition at line 119 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.8 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I3_[3 ∗17] [private]

Fourth interleaving register.

Definition at line 120 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.9 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I4_[4 ∗17] [private]

Fifth interleaving register.

Definition at line 121 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.10 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I5_[5 ∗17] [private]

Sixth interleaving register.

Definition at line 122 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.11 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I6_[6 *17] [private]

Seventh interleaving register.

Definition at line 123 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.12 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I7_[7 *17] [private]

Eighth interleaving register.

Definition at line 124 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.13 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I8_[8 *17] [private]

Ninth interleaving register.

Definition at line 125 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.14 uint8_t iris::phy::Dvbt1ConvInterleaverComponent::I9_[9 *17] [private]

Tenth interleaving register.

Definition at line 126 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.15 int iris::phy::Dvbt1ConvInterleaverComponent::rsOffset_ [private]

Input offset.

Definition at line 129 of file Dvbt1ConvInterleaverComponent.h.

Referenced by process(), and setup().

7.3.5.16 double iris::phy::Dvbt1ConvInterleaverComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 114 of file Dvbt1ConvInterleaverComponent.h.

7.3.5.17 double iris::phy::Dvbt1ConvInterleaverComponent::timeStamp_ [private]

Timestamp of current frame.

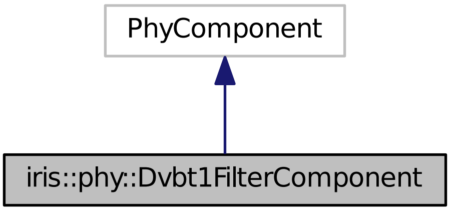
Definition at line 113 of file Dvbt1ConvInterleaverComponent.h.

7.4 iris::phy::Dvbt1FilterComponent Class Reference

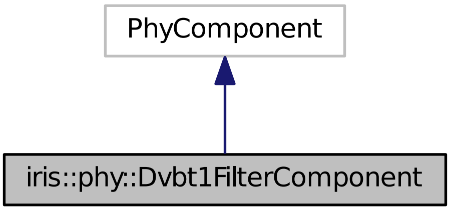
A DVB-T1 filter component.

```
#include <Dvbt1FilterComponent.h>
```

Inheritance diagram for iris::phy::Dvbt1FilterComponent:



Collaboration diagram for iris::phy::Dvbt1FilterComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.
- typedef std::complex< float > Cplx

  A complex type.
- typedef std::vector< Cplx > CplxVec

  A vector of complex.
- typedef CplxVec::iterator CplxVecIt
- typedef std::vector< float > FloatVec

  A vector of float.
- typedef FloatVec::iterator FloatVecIt

  An iterator for a vector of float.
- typedef std::vector< int > IntVec

  A vector of integers.
- typedef IntVec::iterator IntVecIt

  An iterator for a vector of integers.

Public Member Functions

- Dvbt1FilterComponent (std::string name)

  Default constructor.
- ∼Dvbt1FilterComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the mapper ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all our index vectors and containers.
- void destroy ()

  Destroy the component.
- int kaiser_design (int ∗order, double ∗beta, double ripple, double width)

  Find Kaiser parameters.
- int filter_design (FloatVec &h, int order, double fc)

  Design a Kaiser-windowed low-pass filter.
- double kaiser_window (int n, int order, double beta)

  Find Kaiser window coefficients.
- double sinc (double x)

  sin(x)/x function
- double factorial (int n)

  factorial function
- double bessel_I0 (double x)

  Zeroth Order Modified Bessel Function.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double sampleRate_x

  Sampling rate (default = 0)
- double stopBand_x

  Filter stop-band (default = 4000000)
- double sBAttenuation_x

  Filter stop-band attenuation (default = 35)
- std::string coeffsFile_x

  Text file with impulse response (default = none)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- bool symmetric_
- int filterLength_
- FloatVec coeffp_
- CplxVec work_

7.4.1 Detailed Description

A DVB-T1 filter component.

Dvbt1FilterComponent is the second optional block composing the DVB-T transmission chain. It is required only if the spectrum emission mask (SEM) has to be obeyed directly at the BB level and cannot be modified operating on the RF emitted signal. This filter also helps to reduce the IF images resulting from the interpolation process, if the DAC sampling rate is not directly compatible with the DVB-T sampling rate.

This block implements a Kaiser-designed FIR lowpass filter, whose number of taps is decided by the attenuation and transition bandwith values. Please note that setting high values of attenuation or a steep transition bandwidth could result in a high number of taps, and the filter could not be able to operate in real time.

This block accepts in input complex float values and generates in output complex float values.

There are parameters several that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *samplerate*: by default set to "0", a placeholder for 64e6/7 Hz. This represents the sampling rate of the DAC signal and, consequently, the whole bandwidth over which the filter may operate.
- *stopband*: by default set to "4000000.0", it represents the frequency at which the specified attenuation is achieved. This frequency is given relatively to the centre frequency of the RF emitted signal. The transition bandwidth of the filter ends at this frequency, and it begins right after the last active OFDM carrier, which happens to be at 3.805 MHz for an 8K system.
- *attenuation*: by default set to "35.0", it is the attenuation (in dB) of the filter at the specified stop frequency.
- *coeffsfile*: by default set to "", which means not enabled, this is the name of a text file where the impulse response of the filter is saved, line after line.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

Definition at line 90 of file Dvbt1FilterComponent.h.

7.4.2 Member Typedef Documentation

7.4.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1FilterComponent::ByteVec

A vector of bytes.

Definition at line 96 of file Dvbt1FilterComponent.h.

7.4.2.2 typedef ByteVec::iterator iris::phy::Dvbt1FilterComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 99 of file Dvbt1FilterComponent.h.

7.4.2.3 typedef std::complex<float> iris::phy::Dvbt1FilterComponent::Cplx

A complex type.

Definition at line 102 of file Dvbt1FilterComponent.h.

7.4.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1FilterComponent::CplxVec

A vector of complex.

Definition at line 105 of file Dvbt1FilterComponent.h.

7.4.2.5 typedef CplxVec::iterator iris::phy::Dvbt1FilterComponent::CplxVecIt

Definition at line 108 of file Dvbt1FilterComponent.h.

7.4.2.6 typedef std::vector<float> iris::phy::Dvbt1FilterComponent::FloatVec

A vector of float.

Definition at line 111 of file Dvbt1FilterComponent.h.

7.4.2.7 typedef FloatVec::iterator iris::phy::Dvbt1FilterComponent::FloatVecIt

An iterator for a vector of float.

Definition at line 114 of file Dvbt1FilterComponent.h.

7.4.2.8 typedef std::vector<int> iris::phy::Dvbt1FilterComponent::IntVec

A vector of integers.

Definition at line 117 of file Dvbt1FilterComponent.h.

7.4.2.9 typedef IntVec::iterator iris::phy::Dvbt1FilterComponent::IntVecIt

An iterator for a vector of integers.

Definition at line 120 of file Dvbt1FilterComponent.h.

7.4.3 Constructor & Destructor Documentation

7.4.3.1 iris::phy::Dvbt1FilterComponent::Dvbt1FilterComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1FilterComponent.cpp.

References coeffsFile_x, debug_x, sampleRate_x, sBAttenuation_x, and stopBand_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1filter",                    // component type
                "A DVB-T1 filter component",      // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  registerParameter(
    "samplerate", "Sampling rate (use 0 for 9142857)",
    "0.0", true, sampleRate_x, Interval<double>(0.0,15000000.0));

  registerParameter(
    "stopband", "Stop-band of the filter, in Hz relative to the "
    "centre frequency", "4000000.0", true, stopBand_x,
    Interval<double>(2000000.0,10000000.0));

  registerParameter(
    "attenuation", "Attenuation in the stop-band, in dB: 0 disables "
    "filtering (35 is the value tested at Electrosys)", "35.0", true,
    sBAttenuation_x, Interval<double>(0.0,90.0));

  registerParameter(
    "coeffsfile", "Text file with the filter impulse response",
    "", true, coeffsFile_x);
}
```

7.4.3.2 iris::phy::Dvbt1FilterComponent::∼Dvbt1FilterComponent ( )

Default destructor.

Just calls destroy().

Definition at line 92 of file Dvbt1FilterComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.4.4 Member Function Documentation

7.4.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FilterComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 159 of file Dvbt1FilterComponent.h.

```
159 { return &arr[0]; }
```

7.4.4.2 double iris::phy::Dvbt1FilterComponent::bessel_I0 (double *x*) [private]

Zeroth Order Modified Bessel Function.

Parameters

x	The input value

Returns

The function evaluated on x

Definition at line 254 of file Dvbt1FilterComponent.cpp.

References factorial().

Referenced by kaiser_window().

```
{
    double I0 = 1.0;
    int i = 0;
    for (i = 1; i <= 20; i++) {
        I0 += pow(x / 2.0, (double) (2 * i)) / pow(factorial(i), 2.0);
    }
    return I0;
}
```

7.4.4.3 void iris::phy::Dvbt1FilterComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide complex values.

Definition at line 110 of file Dvbt1FilterComponent.cpp.

```
{
  outputTypes["output1"] = TypeInfo< Cplx >::identifier;
}
```

7.4.4.4 void iris::phy::Dvbt1FilterComponent::destroy ( ) [private]

Destroy the component.

Definition at line 443 of file Dvbt1FilterComponent.cpp.

Referenced by parameterHasChanged(), and ~Dvbt1FilterComponent().

```
{
}
```

7.4.4.5 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FilterComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 161 of file Dvbt1FilterComponent.h.

```
{ return &arr[0]+N; }
```

7.4.4.6 double iris::phy::Dvbt1FilterComponent::factorial (int *n*) [private]

factorial function

Parameters

n	the integer on which to apply the factorial

Returns

the factorial of n

Definition at line 241 of file Dvbt1FilterComponent.cpp.

Referenced by bessel_I0().

```
{
    double fact = 1.0;
    int i = 0;
    for (i = 1; i <= n; i++)
        fact *= (double) i;
    return fact;
}
```

7.4.4.7 int iris::phy::Dvbt1FilterComponent::filter_design (**FloatVec** & *h,* int *order,* double *fc*) [private]

Design a Kaiser-windowed low-pass filter.

Implementation inspired from http://www.labbookpages.co.uk/audio/firWindowing.html

Parameters

fc	The cut frequency is normalized to the sampling frequency
order	Order of the filter (as calculated by kaiser_design)
h	The array of filter taps

Returns

0 in case of success

Definition at line 304 of file Dvbt1FilterComponent.cpp.

References sinc().

Referenced by setup().

```
{
    h.resize(order + 1);
    for (int i = 0; i <= order; i++) {
        h[i] = (float) (2.0 * fc * sinc(2.0 * M_PI * fc * ((double) i
          - (double) order / 2.0)));
    }

    return 0;
}
```

7.4.4.8 void iris::phy::Dvbt1FilterComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 120 of file Dvbt1FilterComponent.cpp.

References setup().

```
{
  setup();
}
```

7.4.4.9 int iris::phy::Dvbt1FilterComponent::kaiser_design (int * *order*, double * *beta*, double *ripple*, double *width*) [private]

Find Kaiser parameters.

Implementation inspired from http://www.labbookpages.co.uk/audio/firWindowing.html

Parameters

ripple	Ripple of the filter (linear)
width	Bandwidth normalized to sampling frequency
beta	The β of the Kaiser window
order	The order of the filter (number of taps minus one)

Returns

0 in case of success

Definition at line 274 of file Dvbt1FilterComponent.cpp.

Referenced by setup().

```
{
    double A = -20.0 * log(ripple) / log(10.0);
    double tw = 2.0 * M_PI * width;

    if (A > 21.0)
        *order = (int) ceil((A - 7.95) / (2.285 * tw));
    else
        *order = (int) ceil(5.79 / tw);

    if (A <= 21.0)
        *beta = 0.0;
    else if (21.0 < A && A <= 50.0)
        *beta = 0.5842 * pow(A - 21.0, 0.4) + 0.07886 * (A - 21.0);
    else
        *beta = 0.1102 * (A - 8.7);

    return 0;
}
```

7.4.4.10 double iris::phy::Dvbt1FilterComponent::kaiser_window (int *n*, int *order*, double *beta*) [private]

Find Kaiser window coefficients.

Inspired from http://www.labbookpages.co.uk/audio/firWindowing.html

Parameters

n	The lag at which to evaluate the Kaiser function
beta	The β of the Kaiser window
order	The order of the filter (number of taps minus one)

Returns

The amplitude of the filter tap

Definition at line 323 of file Dvbt1FilterComponent.cpp.

References bessel_I0().

Referenced by setup().

```
{
    return bessel_I0(beta * sqrt(1.0 - pow(((double) (2 * n) / (double) order)
      - 1.0, 2.0))) / bessel_I0(beta);
}
```

7.4.4.11 void iris::phy::Dvbt1FilterComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has several significant parameters

Definition at line 219 of file Dvbt1FilterComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "stopband" || name == "attenuation")
  {
    destroy();
    setup();
  }
}
```

7.4.4.12 void iris::phy::Dvbt1FilterComponent::process ( ) [virtual]

Main processing method.

Definition at line 126 of file Dvbt1FilterComponent.cpp.

References coeffp_, debug_x, filterLength_, symmetric_, and work_.

```
{
  // request input
  DataSet< Cplx > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = insize;

  // request output and pre-fill with zeroes
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, insize);
  fill(out->data.begin(), out->data.end(), Cplx(0,0));

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // copy head
  CplxVecIt workit = work_.begin() + filterLength_ - 1;
  copy(in->data.begin(), in->data.begin() + filterLength_ - 1, workit);

  // filter!
  CplxVecIt outit = out->data.begin();
    if(symmetric_)
    {
        // symmetric filter
        for(int n = 0; n < filterLength_ - 1; n++, outit++)
        {
            CplxVecIt init = workit + n;
            CplxVecIt inlastit = workit + n - filterLength_ + 1;
      FloatVecIt coeffit = coeffp_.begin();
            for(; init > inlastit; init--, inlastit++, coeffit++)
            {
                outit->real(outit->real() + *coeffit * (init->real() + inlastit->real()));
                outit->imag(outit->imag() + *coeffit * (init->imag() + inlastit->imag()));
            }
            outit->real(outit->real() + *coeffit * init->real());
            outit->imag(outit->imag() + *coeffit * init->imag());
        }
        for(int n = filterLength_ - 1; n < insize; n++, outit++)
        {
            CplxVecIt init = in->data.begin() + n;
            CplxVecIt inlastit = in->data.begin() + n - filterLength_ + 1;
      FloatVecIt coeffit = coeffp_.begin();
            for(; init > inlastit; init--, inlastit++, coeffit++)
            {
                outit->real(outit->real() + *coeffit * (init->real() + inlastit->real()));
                outit->imag(outit->imag() + *coeffit * (init->imag() + inlastit->imag()));
            }
            outit->real(outit->real() + *coeffit * init->real());
            outit->imag(outit->imag() + *coeffit * init->imag());
        }
    } else {
        // asymmetric filter - double work
        for(int n = 0; n < filterLength_ - 1; n++, outit++)
        {
            CplxVecIt init = workit + n;
            for(FloatVecIt coeffit = coeffp_.begin(); coeffit <
  coeffp_.end();
              coeffit++, init--)
            {
                outit->real(outit->real() + *coeffit * init->real());
                outit->imag(outit->imag() + *coeffit * init->imag());
            }
        }
        for(int n = filterLength_ - 1; n < insize; n++, outit++)
        {
            CplxVecIt init = in->data.begin() + n;
            for(FloatVecIt coeffit = coeffp_.begin(); coeffit <
  coeffp_.end();
              coeffit++, init--)
            {
                outit->real(outit->real() + *coeffit * init->real());
                outit->imag(outit->imag() + *coeffit * init->imag());
            }
        }
    }

    // copy tail in previous
    copy(in->data.end() - (filterLength_ - 1), in->data.end(), work_.begin());

  // Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseOutputDataSet("output1", out);
  releaseInputDataSet("input1", in);
}
```

7.4.4.13 void iris::phy::Dvbt1FilterComponent::registerPorts ( ) [virtual]

Register the mapper ports with the IRIS system.

This component has one input that accept complex float values and one output that provides complex float values.

Definition at line 101 of file Dvbt1FilterComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< Cplx >::identifier);
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.4.4.14 void iris::phy::Dvbt1FilterComponent::setup ( ) [private]

Set up all our index vectors and containers.

Definition at line 336 of file Dvbt1FilterComponent.cpp.

References coeffp_, coeffsFile_x, filter_design(), filterLength_, kaiser_design(), kaiser_window(), MAX_FILTER_LENGTH, sampleRate_x, sBAttenuation_x, stopBand_x, symmetric_, and work_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // replace the DVB-T sample rate with its real value
  if(sampleRate_x == 0)
    sampleRate_x = 64.0e6/7.0;

  // clear
  symmetric_ = true;
  filterLength_ = 1;
  coeffp_.resize(1);
  coeffp_[0] = 1;

  // test section, leave disabled
  if(false)
```

```
{
  symmetric_ = true;
  filterLength_ = 123;
  coeffp_.resize(filterLength_);
  for(int i = 0; i < filterLength_; i++)
    coeffp_[i] = ((double) i / 4.0)/filterLength_;
}

  // design the transmission filter if requested
  if(true && sBAttenuation_x)
  {
      // checks (you can try to modify the limits, but long filters could result
      if(stopBand_x < 0.515 * (64.0e6/7.0) * (1705.0 / 2048.0))
          LOG(LERROR) << "The selected stopband is too next to the passband: "
            << stopBand_x;
      if(stopBand_x > 0.485 * sampleRate_x)
          LOG(LERROR) << "The selected stopband is too next to the sampling band: "
            << sampleRate_x;
      if(sBAttenuation_x > 40)
          LOG(LERROR) << "A maximum attenuation of 40 dB can be specified";
      if (sBAttenuation_x < 5)
          LOG(LERROR) << "A minimum attenuation of 5 dB can be specified";

      // the transition width is between the last carrier edge and the stopband
      double tw = stopBand_x - 0.5 * (64.0e6/7.0) * (1705.0 / 2048.0);

      // the cutoff frequency is at the last carrier edge plus half transition width
      double fc = 0.501 * (64.0e6/7.0) * (1705.0 / 2048.0) + tw / 2;

      // normalize to sample frequency
      tw /= sampleRate_x;
      fc /= sampleRate_x;

      // the ripple
      double ripple = pow(10.0, - sBAttenuation_x / 20.0);

      // find kaiser parameters
      double beta = 0.0;
      int order = 0;
      int status = kaiser_design(&order, &beta, ripple, tw);
      if(status)
          LOG(LERROR) << "Could not design the Kaiser window";

      // ensure an integer-delay filter is designed (odd length)
      filterLength_ = (2 *((order + 1) / 2)) + 1;

      // check
      if(filterLength_ > MAX_FILTER_LENGTH)
          LOG(LERROR) << "The maximum filter length has been exceeded: relax the "
            "filtering performance";

      // design base filter
      status = filter_design(coeffp_, filterLength_ - 1, fc);
      if(status)
          LOG(LERROR) << "Could not design the base filter";

      // windowed filter
      for(int m = 0; m < filterLength_; m++)
          coeffp_[m] *= (float) kaiser_window(m, filterLength_ - 1, beta);

      // dump filter coefficients to file
    if(!coeffsFile_x.empty())
  {
        FILE *fp = fopen(coeffsFile_x.c_str(), "wt");
        if(fp)
        {
      setlocale(LC_NUMERIC, "C");
            for(int m = 0; m < filterLength_; m++)
                fprintf(fp, "%.8f\n", coeffp_[m]);
            fclose(fp);
        }
      }

      // discover if the filter is symmetric or asymmetric
      // This isn't really needed, since the filter will always be symmetrical
      double maxtol = 1.0E-8, tol = 0.0;
      for (int m = 0; m < filterLength_ / 2; m++)
          tol += fabs(coeffp_[m] - coeffp_[filterLength_ - 1 - m]);
      if (tol < maxtol)
          symmetric_ = true;
      else
          symmetric_ = false;
  }

  // resume filter characteristics
  LOG(LINFO) << (symmetric_ ? "Symmetric" : "Asymmetric") << " filter, "
    << filterLength_ << " taps";

  // working initial array
  work_.resize(filterLength_ - 1 + filterLength_ - 1);
}
```

7.4.4.15 double iris::phy::Dvbt1FilterComponent::sinc (double *x*) [private]

sin(x)/x function

Parameters

x	Input value

Returns

The sinc of the input

Definition at line 232 of file Dvbt1FilterComponent.cpp.

Referenced by filter_design().

```
{
    return x == 0.0 ? 1.0 : (sin(x) / x);
}
```

7.4.5 Member Data Documentation

7.4.5.1 FloatVec iris::phy::Dvbt1FilterComponent::coeffp_ [private]

Definition at line 147 of file Dvbt1FilterComponent.h.

Referenced by process(), and setup().

7.4.5.2 std::string iris::phy::Dvbt1FilterComponent::coeffsFile_x [private]

Text file with impulse response (default = none)

Definition at line 138 of file Dvbt1FilterComponent.h.

Referenced by Dvbt1FilterComponent(), and setup().

7.4.5.3 bool iris::phy::Dvbt1FilterComponent::debug_x [private]

Debug flag (default = false)

Definition at line 134 of file Dvbt1FilterComponent.h.

Referenced by Dvbt1FilterComponent(), and process().

7.4.5.4 int iris::phy::Dvbt1FilterComponent::filterLength_ [private]

Definition at line 146 of file Dvbt1FilterComponent.h.

Referenced by process(), and setup().

7.4.5.5 double iris::phy::Dvbt1FilterComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 144 of file Dvbt1FilterComponent.h.

7.4.5.6 double iris::phy::Dvbt1FilterComponent::sampleRate_x [private]

Sampling rate (default = 0)

Definition at line 135 of file Dvbt1FilterComponent.h.

Referenced by Dvbt1FilterComponent(), and setup().

7.4.5.7 double iris::phy::Dvbt1FilterComponent::sBAttenuation_x [private]

Filter stop-band attenuation (default = 35)

Definition at line 137 of file Dvbt1FilterComponent.h.

Referenced by Dvbt1FilterComponent(), and setup().

7.4.5.8 double iris::phy::Dvbt1FilterComponent::stopBand_x [private]

Filter stop-band (default = 4000000)

Definition at line 136 of file Dvbt1FilterComponent.h.

Referenced by Dvbt1FilterComponent(), and setup().

7.4.5.9 bool iris::phy::Dvbt1FilterComponent::symmetric_ [private]

Definition at line 145 of file Dvbt1FilterComponent.h.

Referenced by process(), and setup().

7.4.5.10 double iris::phy::Dvbt1FilterComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 143 of file Dvbt1FilterComponent.h.

7.4.5.11 CplxVec iris::phy::Dvbt1FilterComponent::work_ [private]

Definition at line 148 of file Dvbt1FilterComponent.h.

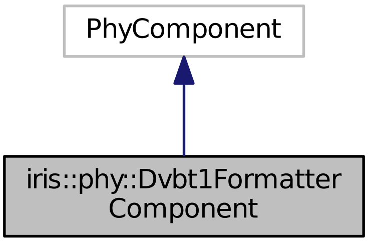
Referenced by process(), and setup().

7.5 iris::phy::Dvbt1FormatterComponent Class Reference

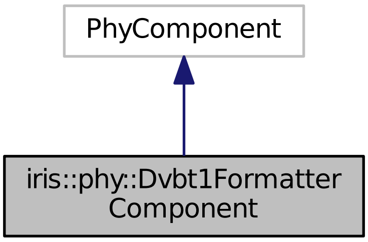
A DVB-T1 formatter component.

`#include <Dvbt1FormatterComponent.h>`

Inheritance diagram for iris::phy::Dvbt1FormatterComponent:



Collaboration diagram for iris::phy::Dvbt1FormatterComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec
- typedef ByteVec::iterator ByteVecIt
- typedef std::complex< float > Cplx
- typedef std::vector< Cplx > CplxVec
- typedef CplxVec::iterator CplxVecIt
- typedef std::vector< int16_t > ShortVec
- typedef ShortVec::iterator ShortVecIt

Public Member Functions

- Dvbt1FormatterComponent (std::string name)
- ∼Dvbt1FormatterComponent ()
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)
- virtual void registerPorts ()
- virtual void initialize ()
- virtual void process ()
- virtual void parameterHasChanged (std::string name)

Private Member Functions

- void setup ()

  Set up all our index vectors and containers.
- void destroy ()

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.

7.5.1 Detailed Description

A DVB-T1 formatter component.

Definition at line 48 of file Dvbt1FormatterComponent.h.

7.5.2 Member Typedef Documentation

7.5.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1FormatterComponent::ByteVec

Definition at line 53 of file Dvbt1FormatterComponent.h.

7.5.2.2 typedef ByteVec::iterator iris::phy::Dvbt1FormatterComponent::ByteVecIt

Definition at line 54 of file Dvbt1FormatterComponent.h.

7.5.2.3 typedef std::complex<float> iris::phy::Dvbt1FormatterComponent::Cplx

Definition at line 55 of file Dvbt1FormatterComponent.h.

7.5.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1FormatterComponent::CplxVec

Definition at line 56 of file Dvbt1FormatterComponent.h.

7.5.2.5 typedef CplxVec::iterator iris::phy::Dvbt1FormatterComponent::CplxVecIt

Definition at line 57 of file Dvbt1FormatterComponent.h.

7.5.2.6 typedef std::vector<int16_t> iris::phy::Dvbt1FormatterComponent::ShortVec

Definition at line 58 of file Dvbt1FormatterComponent.h.

7.5.2.7 typedef ShortVec::iterator iris::phy::Dvbt1FormatterComponent::ShortVecIt

Definition at line 59 of file Dvbt1FormatterComponent.h.

7.5.3 Constructor & Destructor Documentation

7.5.3.1 iris::phy::Dvbt1FormatterComponent::Dvbt1FormatterComponent (std::string *name*)

Definition at line 55 of file Dvbt1FormatterComponent.cpp.

References debug_x.

```
  : PhyComponent(name,                         // component name
                 "dvbt1formatter",             // component type
                 "A DVB-T1 formatter component", // description
                 "Giuseppe Baruffa",           // author
                 "0.1")                        // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);
}
```

7.5.3.2 iris::phy::Dvbt1FormatterComponent::∼Dvbt1FormatterComponent ( )

Definition at line 69 of file Dvbt1FormatterComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.5.4 Member Function Documentation

7.5.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FormatterComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 82 of file Dvbt1FormatterComponent.h.

```
{ return &arr[0]; }
```

7.5.4.2 void iris::phy::Dvbt1FormatterComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Definition at line 80 of file Dvbt1FormatterComponent.cpp.

```
{
  outputTypes["output1"] = TypeInfo< int16_t >::identifier;
}
```

7.5.4.3 void iris::phy::Dvbt1FormatterComponent::destroy ( ) [private]

Definition at line 145 of file Dvbt1FormatterComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1FormatterComponent().

```
{
}
```

7.5.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FormatterComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 84 of file Dvbt1FormatterComponent.h.

```
{ return &arr[0]+N; }
```

7.5.4.5 void iris::phy::Dvbt1FormatterComponent::initialize ( ) [virtual]

Definition at line 87 of file Dvbt1FormatterComponent.cpp.

References setup().

```
{
  setup();
}
```

7.5.4.6 void iris::phy::Dvbt1FormatterComponent::parameterHasChanged (std::string *name*) [virtual]

Definition at line 131 of file Dvbt1FormatterComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "???")
  {
    destroy();
    setup();
  }
}
```

7.5.4.7 void iris::phy::Dvbt1FormatterComponent::process ( ) [virtual]

Definition at line 92 of file Dvbt1FormatterComponent.cpp.

References debug_x.

```
{
  DataSet< Cplx >* in = NULL;
  getInputDataSet("input1", in);
  int insize = in ? (int) in->data.size() : 0;
  int outsize = 2 * insize;

  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  DataSet< int16_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // do the formatting
  ShortVecIt outit = out->data.begin();
  for(CplxVecIt init = in->data.begin(); init < in->data.end(); init++)
  {
    if(init->real() > 1.0)
```

```
      *outit++ = 32767;
    else if(init->real() < -1)
      *outit++ = -32768;
    else
      *outit++ = (int16_t) (0.5 + init->real() * 32768.0);
    if(init->imag() > 1.0)
      *outit++ = 32767;
    else if(init->imag() < -1)
      *outit++ = -32768;
    else
      *outit++ = (int16_t) (0.5 + init->imag() * 32768.0);
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.5.4.8 void iris::phy::Dvbt1FormatterComponent::registerPorts ( ) [virtual]

Definition at line 74 of file Dvbt1FormatterComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< Cplx >::identifier);
  registerOutputPort("output1", TypeInfo< int16_t >::identifier);
}
```

7.5.4.9 void iris::phy::Dvbt1FormatterComponent::setup ( ) [private]

Set up all our index vectors and containers.

Definition at line 141 of file Dvbt1FormatterComponent.cpp.

Referenced by initialize(), and parameterHasChanged().

```
{
}
```

7.5.5 Member Data Documentation

7.5.5.1 bool iris::phy::Dvbt1FormatterComponent::debug_x [private]

Debug flag (default = false)

Definition at line 73 of file Dvbt1FormatterComponent.h.

Referenced by Dvbt1FormatterComponent(), and process().

7.5.5.2 double iris::phy::Dvbt1FormatterComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 79 of file Dvbt1FormatterComponent.h.

7.5.5.3 double iris::phy::Dvbt1FormatterComponent::timeStamp_ [private]

Timestamp of current frame.

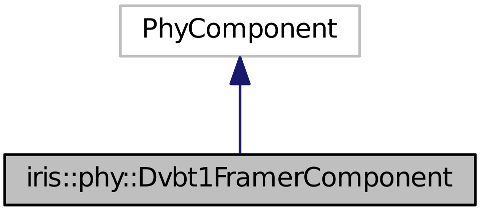
Definition at line 78 of file Dvbt1FormatterComponent.h.

7.6 iris::phy::Dvbt1FramerComponent Class Reference

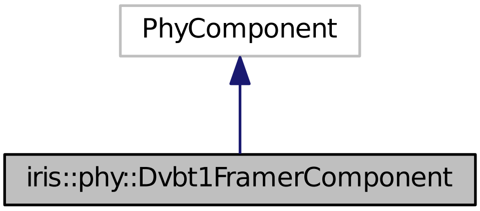
A DVB-T1 framer component.

```
#include <Dvbt1FramerComponent.h>
```

Inheritance diagram for iris::phy::Dvbt1FramerComponent:



Collaboration diagram for iris::phy::Dvbt1FramerComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.
- typedef std::complex< float > Cplx

  A complex type.
- typedef std::vector< Cplx > CplxVec

  A vector of complex.
- typedef CplxVec::iterator CplxVecIt

Public Member Functions

- Dvbt1FramerComponent (std::string name)

Default constructor.

- ∼Dvbt1FramerComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the mapper ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all needed constants.
- void destroy ()

  Destroy the component.
- int t1_tps_generate (unsigned char ∗tps, int block_in_frame, int frame_in_superframe)

  This functions generates the modulated TPS carriers.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int ofdmMode_x

  OFDM mode (default = 2048)
- int cellId_x

  Cell ID for DVB-H mode (default = -1)
- int qamMapping_x

  QAM constellation mapping (default = 16)
- bool inDepthInterleaver_x

  In-depth interleaver for DVB-H mode (default = false)
- int hyerarchyMode_x

  Hyerarchical mode (default = 0)
- int hpCodeRate_x

  HP stream channel coding rate (default = 34)
- int lpCodeRate_x

  LP stream channel coding rate (default = 34)

- int deltaMode_x

  Cyclic prefix ratio (default = 32)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int nMax_

  data carriers
- int kMax_

  active carriers
- int fraOffset_

  framer offset
- CplxVec fraRegister_

  framer template
- int blockIndex_

  OFDM block index.
- float tpsAmpl_ [6817]

  tps_amplitudes
- uint8_t tps_ [T1_BLOCKS_PER_FRAME]

  tps data

Static Private Attributes

- static int cont_pilot_position [178]
- static int tps_position [69]
- static unsigned char prbs_pilot [6817]

7.6.1 Detailed Description

A DVB-T1 framer component.

Dvbt1FramerComponent is the ninth block composing the DVB-T transmission chain. The framer has the task to assemble together QAM data cells, pilot data cells, and transmission parameters signaling (TPS) data cells into a frame structure that will be mapped onto OFDM symbols.

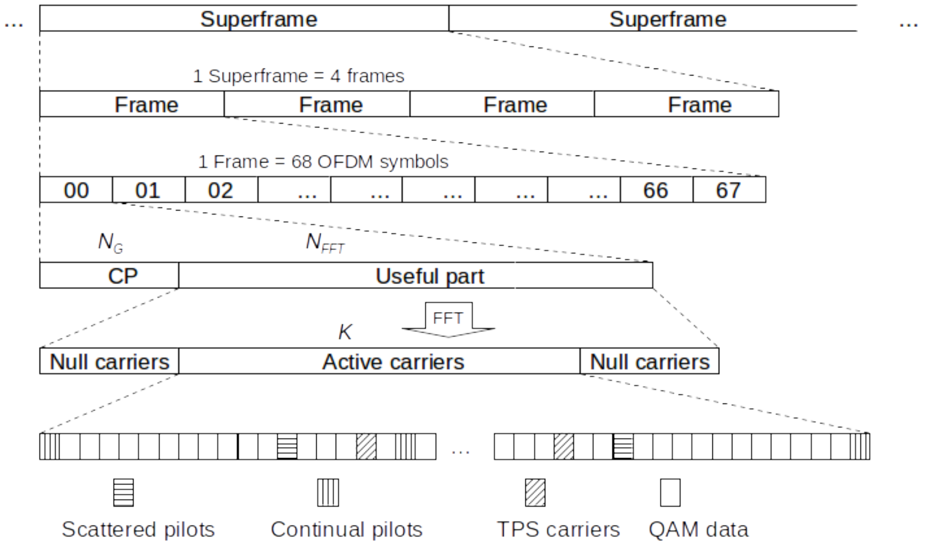


Figure 7.4: DVB-T framing structure.

The basic frame structure starts from the OFDM symbol: 68 OFDM symbols constitute one frame, and 4 frames build up a superframe. Each OFDM symbol is composed by an useful portion, which comes from an IFFT operation, and by a cyclic prefix (CP). The carriers of the useful portion are composed by active and null carriers, which are switched off and are guard bands.

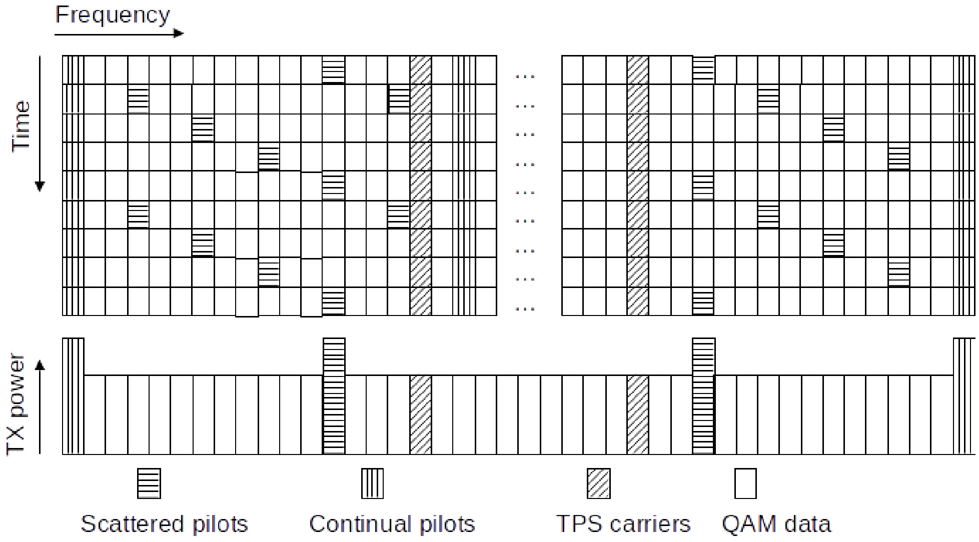


Figure 7.5: DVB-T pilots structure.

Pilot carriers are divided between continual pilots, which occur on every OFDM symbol at the same carrier position, and scattered pilots, which are cyclically shifted of three positions at each new OFDM symbol. Additionally, there are a number of carriers that are used to convey TPS data, useful for purposes of frame synchronization and signalling. As displayed in the figure above, the carriers are not created at the same power: while data and TPS carriers have a unitary power, all pilot carriers are transmitted at a power of 16/9.

This block accepts in input complex float values and generates in output complex float values. The block is capable to generate internally all the required frame timing and modulation for the pilot and TPS cells.

There are several parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *hpcoderate*: by default set to "34", this is used to select one of the five possible coding rates. The admitted values are "12", "23", "34", "56", and "78", which are easily recognizable as the real coding ratioes written without the separating slash. This parameter refers to the high priority (HP) stream in case of hyerarchical transmission, differently it refers to the coderate of the single stream for nonhyerarchical transmission.
- *lpcoderate*: by default set to "34", this is used to select one of the five possible coding rates. The admitted values are "12", "23", "34", "56", and "78", which are easily recognizable as the real coding ratioes written without the separating slash. This parameter refers to the low priority (LP) stream in case of hyerarchical transmission, differently it is not used for nonhyerarchical transmission.
- *qammapping*: by default set to "16", this is used to select one of the three possible QAM mappings. The admitted values are "4", "16", "64".
- *hyerarchymode*: by default set to "0", which means "not hyerarchical". Hierarchical modes are used to transmit two different transport streams, one with a high priority (HP) information and another one with a low priority (LP) information. The admitted values are "0, "1", "2", "4". NOTE: hyerarchical modes are not implemented in the current release of this modulator.
- *ofdmmode*: by default set to "2048", this is used to select one of the three possible OFDM modes. The admitted values are "2048", "4096", "8192", respectively for 2K, 4K (DVB-H, unused), and 8K.

- *deltamode*: by default set to "32", this is used to select one of the four possible cyclic prefix lengths. The admitted values are "32", "16", "8", and "4", which are directly derived from the denominator of the cyclic prefix fraction (1/32, 1/16, 1/8, 1/4).
- *cellid*: by default set to "-1", which means it is disabled. The Cell Identifier is used to identify transmission towers with a 16 bit numeric identifier, and is used only in case of DVB-H transmission. NOTE: DVB-H is not implemented in this software modulator.
- *indepthinterleaver*: by default set to "false", which means it is disabled. This additional interleaver is used only in DVB-H mode and should improve the diversity of the received signal in case of transmission over time varying channels. NOTE: DVB-H is not implemented in this software modulator.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

Definition at line 132 of file Dvbt1FramerComponent.h.

7.6.2 Member Typedef Documentation

7.6.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1FramerComponent::ByteVec

A vector of bytes.

Definition at line 138 of file Dvbt1FramerComponent.h.

7.6.2.2 typedef ByteVec::iterator iris::phy::Dvbt1FramerComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 141 of file Dvbt1FramerComponent.h.

7.6.2.3 typedef std::complex<float> iris::phy::Dvbt1FramerComponent::Cplx

A complex type.

Definition at line 144 of file Dvbt1FramerComponent.h.

7.6.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1FramerComponent::CplxVec

A vector of complex.

Definition at line 147 of file Dvbt1FramerComponent.h.

7.6.2.5 typedef CplxVec::iterator iris::phy::Dvbt1FramerComponent::CplxVecIt

Definition at line 150 of file Dvbt1FramerComponent.h.

7.6.3 Constructor & Destructor Documentation

7.6.3.1 iris::phy::Dvbt1FramerComponent::Dvbt1FramerComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1FramerComponent.cpp.

References begin(), cellId_x, debug_x, deltaMode_x, end(), hpCodeRate_x, hyerarchyMode_x, inDepthInterleaver_x, lpCodeRate_x, ofdmMode_x, and qamMapping_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1framer",                    // component type
                "A DVB-T1 framer component",      // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  int codearr[] = {12,23,34,56,78};
  registerParameter(
    "hpcoderate", "HP stream channel coding rate",
    "34", true, hpCodeRate_x, list<int>(begin(codearr),end(codearr)));

  registerParameter(
    "lpcoderate", "LP stream channel coding rate",
    "34", true, lpCodeRate_x, list<int>(begin(codearr),end(codearr)));

  int qamarr[] = {4,16,64};
  registerParameter(
    "qammapping", "QAM constellation mapping",
    "16", true, qamMapping_x, list<int>(begin(qamarr),end(qamarr)));

  int harr[] = {0,1,2,4};
  registerParameter(
    "hyerarchymode", "Hyerarchical mode (0 = NH)",
    "0", true, hyerarchyMode_x, list<int>(begin(harr),end(harr)));

  int ofdmarr[] = {2048,4096,8192};
  registerParameter(
    "ofdmmode", "OFDM mode",
    "2048", true, ofdmMode_x, list<int>(begin(ofdmarr),end(ofdmarr)));

  int deltaarr[] = {32,16,8,4};
  registerParameter(
    "deltamode", "Cyclic prefix ratio",
    "32", true, deltaMode_x, list<int>(begin(deltaarr),end(deltaarr)));

  registerParameter(
    "cellid", "Cell ID for DVB-H mode",
    "-1", true, cellId_x, Interval<int>(-1,65535));

  registerParameter(
    "indepthinterleaver", "In-depth interleaver for DVB-H mode",
    "false", true, inDepthInterleaver_x);
}
```

7.6.3.2 iris::phy::Dvbt1FramerComponent::∼Dvbt1FramerComponent ( )

Default destructor.

Just calls destroy().

Definition at line 111 of file Dvbt1FramerComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.6.4 Member Function Documentation

7.6.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FramerComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 197 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent().

```
197 { return &arr[0]; }
```

7.6.4.2 void iris::phy::Dvbt1FramerComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide complex values.

Definition at line 129 of file Dvbt1FramerComponent.cpp.

```
132 {
133   outputTypes["output1"] = TypeInfo< Cplx >::identifier;
134 }
```

7.6.4.3 void iris::phy::Dvbt1FramerComponent::destroy ( ) [private]

Destroy the component.

Definition at line 943 of file Dvbt1FramerComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1FramerComponent().

```
944 {
945 }
```

7.6.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1FramerComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 199 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent().

```
199 { return &arr[0]+N; }
```

7.6.4.5 void iris::phy::Dvbt1FramerComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 139 of file Dvbt1FramerComponent.cpp.

References setup().

```
140 {
141   setup();
142 }
```

7.6.4.6 void iris::phy::Dvbt1FramerComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has several significant parameters

Definition at line 904 of file Dvbt1FramerComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "hpcoderate" || name == "deltamode" ||
     name == "lpcoderate" || name == "qammapping" ||
     name == "hyerarchymode" || name == "ofdmmode" ||
     name == "cellid" || name == "indepthinterleaver")
  {
    destroy();
    setup();
  }
}
```

7.6.4.7 void iris::phy::Dvbt1FramerComponent::process ( ) [virtual]

Main processing method.

Definition at line 783 of file Dvbt1FramerComponent.cpp.

References blockIndex_, cont_pilot_position, debug_x, fraOffset_, fraRegister_, kMax_, nMax_, prbs_pilot, T1_-BLOCKS_PER_FRAME, T1_FRAMES_PER_SUPERFRAME, T1_PIL_AMPL, T1_TPS_AMPL, t1_tps_generate(), tps_, tps_position, and tpsAmpl_.

```
{
  // request input
  DataSet< Cplx > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = kMax_ * ((insize + fraOffset_) / nMax_);

  // request output
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // fill register
  for(CplxVecIt init = in->data.begin(), outit = out->data.begin();
    init < in->data.end(); init++)
  {
    // copy
    fraRegister_[fraOffset_++] = *init;

    // ready for new block - trigger
    if(fraOffset_ == nMax_)
    {
      // reset offset
      fraOffset_ = 0;

      // initial position values
      int scatt_pil_pos = 3 * (blockIndex_ & 0x03);
      int cpp = 0;
      int cont_pil_pos = cont_pilot_position[cpp];
      int tp = 0;
      int tps_pos = tps_position[tp];

      // counters
      int frameInSuperFrame = blockIndex_ / T1_BLOCKS_PER_FRAME;
      int blockInFrame = blockIndex_ - frameInSuperFrame *
  T1_BLOCKS_PER_FRAME;

      // generate the tps information for this frame
      if(blockInFrame == 0)
      {
        int status = t1_tps_generate(tps_, blockInFrame, frameInSuperFrame);
        if(status)
```

```
          LOG(LERROR) << "Error in TPS parity generation";
      }

      // populate the frame
      CplxVecIt regit = fraRegister_.begin();
          for(int k = 0; k < kMax_; k++, outit++)
          {
              if(k == scatt_pil_pos) {
                  // scattered pilot
                  outit->real(prbs_pilot[k] ? -T1_PIL_AMPL : T1_PIL_AMPL);
                  outit->imag(0);
                  scatt_pil_pos += 12;
                  if (k == cont_pil_pos)
                  {
                      // coincidence with continual pilot
                      cont_pil_pos = cont_pilot_position[++cpp];
                  }
              }
              else if(k == cont_pil_pos)
              {
                  // continual pilot
                  outit->real(prbs_pilot[k] ? -T1_PIL_AMPL : T1_PIL_AMPL);
                  outit->imag(0);
                  cont_pil_pos = cont_pilot_position[++cpp];
              }
              else if(k == tps_pos)
              {
                  // TPS
                  // first symbol in frame: absolute reference for differential encoding
                  if (blockInFrame == 0)
                  {
                      tpsAmpl_[k] = prbs_pilot[k] ? -T1_TPS_AMPL : T1_TPS_AMPL;
                  }
                  else
                  {
                      // subsequent symbols in frame
                      // differentially encoded bits with respect to the first frame bits
                      tpsAmpl_[k] = tps_[blockInFrame] ? -tpsAmpl_[k] : tpsAmpl_[k];
                  }
                  outit->real(tpsAmpl_[k]);
                  outit->imag(0);


                  tps_pos = tps_position[++tp];
              }
              else
              {
                  // real data
                  *outit = *regit++;
              }
          }


          // advance block index
          if(++blockIndex_ == T1_BLOCKS_PER_FRAME * T1_FRAMES_PER_SUPERFRAME)
          {
            // reset
            blockIndex_ = 0;
          }
    }
  }

  // Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.6.4.8 void iris::phy::Dvbt1FramerComponent::registerPorts ( ) [virtual]

Register the mapper ports with the IRIS system.

This component has one input that accept complex float symbols and one output that provides complex float symbols.

Definition at line 120 of file Dvbt1FramerComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< Cplx >::identifier);
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.6.4.9 void iris::phy::Dvbt1FramerComponent::setup ( ) [private]

Set up all needed constants.

Definition at line 917 of file Dvbt1FramerComponent.cpp.

References blockIndex_, fraOffset_, fraRegister_, kMax_, nMax_, ofdmMode_x, tps_, and tpsAmpl_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // prepare
  blockIndex_ = 0;
  fraOffset_ = 0;
  switch(ofdmMode_x)
  {
    case 2048:
      kMax_ = 1705;
      nMax_ = 1512;
      break;
    case 4096:
      kMax_ = 3409;
      nMax_ = 3024;
      break;
    case 8192:
      kMax_ = 6817;
      nMax_ = 6048;
      break;
  }
  fraRegister_.resize(nMax_);
  memset(tpsAmpl_, 0, sizeof tpsAmpl_);
  memset(tps_, 0, sizeof tps_);
}
```

7.6.4.10 int iris::phy::Dvbt1FramerComponent::t1_tps_generate (unsigned char ∗ *tps*, int *block_in_frame*, int *frame_in_superframe*) [private]

This functions generates the modulated TPS carriers.

Parameters

tps	Preallocated array that will contain the bit to be transmitted in the TPS carriers
block_in_frame	Index of the block in the current frame (0...67)
frame_in_superframe	Index of the frame in the current superframe (0...3)

Returns

0 if all went well, else errors happened

Definition at line 390 of file Dvbt1FramerComponent.cpp.

References cellId_x, deltaMode_x, hpCodeRate_x, hyerarchyMode_x, inDepthInterleaver_x, lpCodeRate_x, ofdmMode_x, qamMapping_x, T1_K_BCH, and T1_N_BCH.

Referenced by process().

```
{
    int i = 0, j = 0;
    unsigned char feedback = 0;
    unsigned char x[T1_K_BCH] = {0}, b[T1_N_BCH - T1_K_BCH] = {0};

    // Code generator polynomial
    static unsigned char g[] = {1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1};


    // Predefined TPS bits

    // Very first bit is given by PRBS and it is used
    // as a reference in differential BPSK modulation
    // The present value is set to zero, but it reall
    // doesn't care
    tps[0] = 0;

    // Synchronization
    switch(frame_in_superframe)
  {
        // Frame 1 and 3
    case 0:
    case 2:
        tps[1]  = 0;
        tps[2]  = 0;
        tps[3]  = 1;
        tps[4]  = 1;
        tps[5]  = 0;
        tps[6]  = 1;
        tps[7]  = 0;
        tps[8]  = 1;
        tps[9]  = 1;
        tps[10] = 1;
        tps[11] = 1;
        tps[12] = 0;
        tps[13] = 1;
        tps[14] = 1;
        tps[15] = 1;
        tps[16] = 0;
        break;
        // Frame 2 and 4
    case 1:
    case 3:
        tps[1]  = 1;
        tps[2]  = 1;
        tps[3]  = 0;
        tps[4]  = 0;
        tps[5]  = 1;
        tps[6]  = 0;
        tps[7]  = 1;
        tps[8]  = 0;
        tps[9]  = 0;
        tps[10] = 0;
        tps[11] = 0;
        tps[12] = 1;
        tps[13] = 0;
        tps[14] = 0;
        tps[15] = 0;
        tps[16] = 1;
        break;
    }

    // TPS length indicator
    // Full DVB-H option wants 33 bits
    // NOT USED NOW !!!
    if(cellId_x >= 0)
  {
        // Cell Id is set, 31 bits
        tps[17] = 0;
        tps[18] = 1;
        tps[19] = 1;
        tps[20] = 1;
        tps[21] = 1;
        tps[22] = 1;
    }
  else
  {
        // Cell Id is not set, 23 bits
        tps[17] = 0;
        tps[18] = 1;
        tps[19] = 0;
        tps[20] = 1;
        tps[21] = 1;
        tps[22] = 1;
    }

    // Variable TPS bits

    // Counts the frame number in superframe
    switch(frame_in_superframe)
  {
    case 0:
        tps[23] = 0;
        tps[24] = 0;
        break;
    case 1:
        tps[23] = 0;
        tps[24] = 1;
        break;
    case 2:
        tps[23] = 1;
        tps[24] = 0;
        break;
    case 3:
        tps[23] = 1;
        tps[24] = 1;
    }

    // Constellation
    switch(qamMapping_x)
  {
        // QPSK
    case 4:
        tps[25] = 0;
        tps[26] = 0;
        break;
        // 16-QAM
    case 16:
        tps[25] = 0;
        tps[26] = 1;
        break;
        // 64-QAM
    case 64:
        tps[25] = 1;
        tps[26] = 0;
        break;
    }

    // In-depth inner interleaver
    if(inDepthInterleaver_x == false)
    {
        // Native
        tps[27] = 0;
    }
    else
    {
        tps[27] = 1;
    }

    // Hierarchy
    switch(hyerarchyMode_x)
  {
        // Not hierarchical
    case 0:
        tps[28] = 0;
        tps[29] = 0;
        break;
        // Alpha = 1
    case 1:
        tps[28] = 0;
        tps[29] = 1;
        break;
        // Alpha = 2
    case 2:
        tps[28] = 1;
        tps[29] = 0;
        break;
        // Alpha = 4
    case 4:
        tps[28] = 1;
        tps[29] = 1;
        break;
    }

    // Code rate
    if(hyerarchyMode_x == 0)
  {
        // Code rate, NH
        switch (hpCodeRate_x) {
        case 12:
            tps[30] = 0;
            tps[31] = 0;
            tps[32] = 0;
            break;
        case 23:
            tps[30] = 0;
            tps[31] = 0;
            tps[32] = 1;
```

```
              break;
          case 34:
              tps[30] = 0;
              tps[31] = 1;
              tps[32] = 0;
              break;
          case 56:
              tps[30] = 0;
              tps[31] = 1;
              tps[32] = 1;
              break;
          case 78:
              tps[30] = 1;
              tps[31] = 0;
              tps[32] = 0;
              break;
          default:
              break;
          }
      }
    else
    {
          // Code rate, HP
          switch(hpCodeRate_x)
      {
          case 12:
              tps[30] = 0;
              tps[31] = 0;
              tps[32] = 0;
              break;
          case 23:
              tps[30] = 0;
              tps[31] = 0;
              tps[32] = 1;
              break;
          case 34:
              tps[30] = 0;
              tps[31] = 1;
              tps[32] = 0;
              break;
          case 56:
              tps[30] = 0;
              tps[31] = 1;
              tps[32] = 1;
              break;
          case 78:
              tps[30] = 1;
              tps[31] = 0;
              tps[32] = 0;
              break;
          default:
              break;
          }

          // Code rate, LP
          switch(lpCodeRate_x)
      {
          case 12:
              tps[33] = 0;
              tps[34] = 0;
              tps[35] = 0;
              break;
          case 23:
              tps[33] = 0;
              tps[34] = 0;
              tps[35] = 1;
              break;
          case 34:
              tps[33] = 0;
              tps[34] = 1;
              tps[35] = 0;
              break;
          case 56:
              tps[33] = 0;
              tps[34] = 1;
              tps[35] = 1;
              break;
          case 78:
              tps[33] = 1;
              tps[34] = 0;
              tps[35] = 0;
              break;
          default:
              break;
          }
      }

```

```
  // Guard interval
  switch(deltaMode_x)
{
      case 32:
          tps[36] = 0;
          tps[37] = 0;
          break;
      case 16:
          tps[36] = 0;
          tps[37] = 1;
          break;
      case 8:
          tps[36] = 1;
          tps[37] = 0;
          break;
      case 4:
          tps[36] = 1;
          tps[37] = 1;
          break;
  }

  // Transmission mode
  switch (ofdmMode_x)
{
  case 2048:
      tps[38] = 0;
      tps[39] = 0;
      break;
  case 8192:
      tps[38] = 0;
      tps[39] = 1;
      break;
  case 4096:
      tps[38] = 1;
      tps[39] = 0;
      break;
  }

  // Cell identification bits
  if(cellId_x >= 0)
{
      // Compute Cell Id bits
      unsigned char cidv[16];
      for(i = 0; i < 16; i++)
          cidv[i] = (unsigned char) ((((unsigned short int) cellId_x) >> i) & 0x0001);

      switch(frame_in_superframe)
  {
          // First half in frames 1 and 3
          case (0):
          case (2):
              for (i = 40; i <= 47; i++)
                  tps[i] = cidv[55 - i];
              break;

          // Second half in frames 2 and 4
          case (1):
          case (3):
              for (i = 40; i <= 47; i++)
                  tps[i] = cidv[47 - i];
              break;
      }
  }
else
{
      // Cell Id not set
      for(i = 40; i <= 47; i++)
          tps[i] = 0;
  }

  // Bits 48 and 49 are used in DVB-H
  // Not set currently
  tps[48] = 0;
  tps[49] = 0;

  // Bits from 50 to 53 are all set to zero
  tps[50] = 0;
  tps[51] = 0;
  tps[52] = 0;
  tps[53] = 0;

  // BCH encoding

  // Empty the parity register
  for(i = 0; i < (T1_N_BCH - T1_K_BCH); i++)
      b[i] = 0;

```

```
    // Reverse copy data into x, considering the shortening zeroes
    for(i = 53; i > 0; i--)
        x[53 - i] = tps[i];

    // Compute redundacy bb[], the coefficients of b(x). The redundancy
    // polynomial b(x) is the remainder after dividing x^(n-k)*data(x)
    // by the generator polynomial g(x).
    for(i = (T1_K_BCH - 1); i >= 0; i--)
  {
        feedback = x[i] ^ b[T1_N_BCH - T1_K_BCH - 1];
        if(feedback != 0)
    {
            for(j = (T1_N_BCH - T1_K_BCH - 1); j > 0; j--)

                if (g[j] != 0)
                    b[j] = b[j - 1] ^ feedback;
                else
                    b[j] = b[j - 1];

            b[0] = g[0] && feedback;

        }
    else
    {
            for(j = (T1_N_BCH - T1_K_BCH - 1); j > 0; j--)
                b[j] = b[j - 1];

            b[0] = 0;
        }
    }

    /* Back copy parity bits into s */
    for(i = 0; i < 14; i++)
        tps[54 + i] = b[13 - i];

    return 0;
}
```

7.6.5 Member Data Documentation

7.6.5.1 int iris::phy::Dvbt1FramerComponent::blockIndex_ [private]

OFDM block index.

Definition at line 185 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.2 int iris::phy::Dvbt1FramerComponent::cellId_x [private]

Cell ID for DVB-H mode (default = -1)

Definition at line 166 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.3 int iris::phy::Dvbt1FramerComponent::cont_pilot_position [static], [private]

Initial value:

```
= {
       0,   48,   54,   87,  141,  156,  192,  201,  255,  279,  282,  333,  432,  450,  483,
     525,  531,  618,  636,  714,  759,  765,  780,  804,  873,  888,  918,  939,  942,  969,
     984, 1050, 1101, 1107, 1110, 1137, 1140, 1146, 1206, 1269, 1323, 1377, 1491, 1683, 1704,
    1752, 1758, 1791, 1845, 1860, 1896, 1905, 1959, 1983, 1986, 2037, 2136, 2154, 2187, 2229,
    2235, 2322, 2340, 2418, 2463, 2469, 2484, 2508, 2577, 2592, 2622, 2643, 2646, 2673, 2688,
    2754, 2805, 2811, 2814, 2841, 2844, 2850, 2910, 2973, 3027, 3081, 3195, 3387, 3408, 3456,
    3462, 3495, 3549, 3564, 3600, 3609, 3663, 3687, 3690, 3741, 3840, 3858, 3891, 3933, 3939,
    4026, 4044, 4122, 4167, 4173, 4188, 4212, 4281, 4296, 4326, 4347, 4350, 4377, 4392, 4458,
    4509, 4515, 4518, 4545, 4548, 4554, 4614, 4677, 4731, 4785, 4899, 5091, 5112, 5160, 5166,
    5199, 5253, 5268, 5304, 5313, 5367, 5391, 5394, 5445, 5544, 5562, 5595, 5637, 5643, 5730,
    5748, 5826, 5871, 5877, 5892, 5916, 5985, 6000, 6030, 6051, 6054, 6081, 6096, 6162, 6213,
    6219, 6222, 6249, 6252, 6258, 6318, 6381, 6435, 6489, 6603, 6795, 6816, -1
}
```

Definition at line 189 of file Dvbt1FramerComponent.h.

Referenced by process().

7.6.5.4 bool iris::phy::Dvbt1FramerComponent::debug_x [private]

Debug flag (default = false)

Definition at line 164 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and process().

7.6.5.5 int iris::phy::Dvbt1FramerComponent::deltaMode_x [private]

Cyclic prefix ratio (default = 32)

Definition at line 172 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.6 int iris::phy::Dvbt1FramerComponent::fraOffset_ [private]

framer offset

Definition at line 182 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.7 CplxVec iris::phy::Dvbt1FramerComponent::fraRegister_ [private]

framer template

Definition at line 183 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.8 int iris::phy::Dvbt1FramerComponent::hpCodeRate_x [private]

HP stream channel coding rate (default = 34)

Definition at line 170 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.9 int iris::phy::Dvbt1FramerComponent::hyerarchyMode_x [private]

Hyerarchical mode (default = 0)

Definition at line 169 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.10 bool iris::phy::Dvbt1FramerComponent::inDepthInterleaver_x [private]

In-depth interleaver for DVB-H mode (default = false)

Definition at line 168 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.11 int iris::phy::Dvbt1FramerComponent::kMax_ [private]

active carriers

Definition at line 181 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.12 int iris::phy::Dvbt1FramerComponent::lpCodeRate_x [private]

LP stream channel coding rate (default = 34)

Definition at line 171 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.13 int iris::phy::Dvbt1FramerComponent::nMax_ [private]

data carriers

Definition at line 180 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.14 int iris::phy::Dvbt1FramerComponent::ofdmMode_x [private]

OFDM mode (default = 2048)

Definition at line 165 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), setup(), and t1_tps_generate().

7.6.5.15 unsigned char iris::phy::Dvbt1FramerComponent::prbs_pilot [static], [private]

Definition at line 191 of file Dvbt1FramerComponent.h.

Referenced by process().

7.6.5.16 int iris::phy::Dvbt1FramerComponent::qamMapping_x [private]

QAM constellation mapping (default = 16)

Definition at line 167 of file Dvbt1FramerComponent.h.

Referenced by Dvbt1FramerComponent(), and t1_tps_generate().

7.6.5.17 double iris::phy::Dvbt1FramerComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 178 of file Dvbt1FramerComponent.h.

7.6.5.18 double iris::phy::Dvbt1FramerComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 177 of file Dvbt1FramerComponent.h.

7.6.5.19 uint8_t iris::phy::Dvbt1FramerComponent::tps_[T1_BLOCKS_PER_FRAME] [private]

tps data

Definition at line 187 of file Dvbt1FramerComponent.h.

Referenced by process(), and setup().

7.6.5.20 int iris::phy::Dvbt1FramerComponent::tps_position [static], [private]

Initial value:

```
= {
      34,   50,  209,  346,  413,  569,  595,  688,  790,  901, 1073, 1219, 1262, 1286, 1469,
    1594, 1687, 1738, 1754, 1913, 2050, 2117, 2273, 2299, 2392, 2494, 2605, 2777, 2923, 2966,
    2990, 3173, 3298, 3391, 3442, 3458, 3617, 3754, 3821, 3977, 4003, 4096, 4198, 4309, 4481,
    4627, 4670, 4694, 4877, 5002, 5095, 5146, 5162, 5321, 5458, 5525, 5681, 5707, 5800, 5902,
    6013, 6185, 6331, 6374, 6398, 6581, 6706, 6799, -1
}
```

Definition at line 190 of file Dvbt1FramerComponent.h.

Referenced by process().

7.6.5.21 float iris::phy::Dvbt1FramerComponent::tpsAmpl_[6817] [private]

tps_amplitudes

Definition at line 186 of file Dvbt1FramerComponent.h.

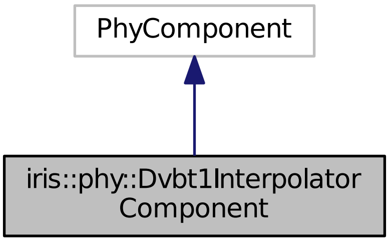
Referenced by process(), and setup().

7.7 iris::phy::Dvbt1InterpolatorComponent Class Reference

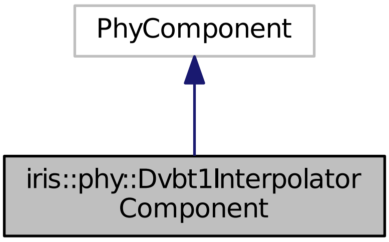
A DVB-T1 interpolator component.

`#include <Dvbt1InterpolatorComponent.h>`

Inheritance diagram for iris::phy::Dvbt1InterpolatorComponent:



Collaboration diagram for iris::phy::Dvbt1InterpolatorComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.
- typedef std::complex< float > Cplx

  A complex type.
- typedef std::vector< Cplx > CplxVec

  A vector of complex.
- typedef CplxVec::iterator CplxVecIt
- typedef std::vector< float > FloatVec

  A vector of float.
- typedef FloatVec::iterator FloatVecIt

  An iterator for a vector of float.
- typedef std::vector< int > IntVec

  A vector of integers.
- typedef IntVec::iterator IntVecIt

  An iterator for a vector of typedef.

Public Member Functions

- Dvbt1InterpolatorComponent (std::string name)

  Default constructor.
- ∼Dvbt1InterpolatorComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the mapper ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

Main processing method.

- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all our index vectors and containers.
- void destroy ()

  Destroy the component.
- int time_buffer_size (int input_samples)

  size correctly the interpolation buffers
- int find_rational_approximation (int ∗num, int ∗den, double x, int N)

  find a rational approximation of a real value
- double ∗ blackman_sinc (int ∗n_order, double T, double dt, int order)

  Calculate a Blackman-windowed sinc.
- double interp_response (double ∗h, int n, double dt, double t)

  interpolate a base response
- double sinc (double x)

  sin(x)/x function

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double inSampleRate_x

  Input sampling rate (default = 0)
- double outSampleRate_x

  Output sampling rate (default = 0)
- std::string responseFile_x

  Text file with impulse response (default = none)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int tiInsize_
- int tiOutsize_
- int inOffset_
- CplxVec inReg_
- int inLength_
- IntVec tiBasepointIndex_
- FloatVec tiHl_

7.7.1 Detailed Description

A DVB-T1 interpolator component.

Dvbt1InterpolatorComponent is the first optional block composing the DVB-T transmission chain. It is required only if the analog conversion module following in the transmission chain has a rate different than that of the natural DVB-T sampling rate (64/7 MHz).

The conversion between the input DVB-T sampling rate and the output sampling rate is performed via a very simple sinc-shaped interpolator. The memory of the interpolating response should be kept short, in order to achieve the best processing speed. Clearly, this block distorts the original signal spectrum, and proper actions should be taken to override this detrimental effect. If you have used the DVB-T OFDM modulator previously on the transmission chain, then this effect has already been taken into account and the generated signal spectrum has been linearly predistorted in order to compensate for the distortion that is generated by the interpolator block.

This block accepts in input complex float values and generates in output complex float values.

There are parameters several that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *insamplerate*: by default set to "0", a placeholder for 64e6/7 Hz. This represents the sampling rate of the entering signal. **Please note that if you are using the Dvbt1OFDM block, then you need to leave this parameter at 0**.
- *outsamplerate*: by default set to "0", a placeholder for 64e6/7 Hz. This represents the sampling rate adopted by the DAC for emitting the BB analog signal
- *responsefile*: by default set to "", which means not enabled, this is the name of a text file where the impulse response of the interpolating filter is saved, line after line.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

Definition at line 90 of file Dvbt1InterpolatorComponent.h.

7.7.2 Member Typedef Documentation

7.7.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1InterpolatorComponent::ByteVec

A vector of bytes.

Definition at line 96 of file Dvbt1InterpolatorComponent.h.

7.7.2.2 typedef ByteVec::iterator iris::phy::Dvbt1InterpolatorComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 99 of file Dvbt1InterpolatorComponent.h.

7.7.2.3 typedef std::complex<float> iris::phy::Dvbt1InterpolatorComponent::Cplx

A complex type.

Definition at line 102 of file Dvbt1InterpolatorComponent.h.

7.7.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1InterpolatorComponent::CplxVec

A vector of complex.

Definition at line 105 of file Dvbt1InterpolatorComponent.h.

7.7.2.5 typedef CplxVec::iterator iris::phy::Dvbt1InterpolatorComponent::CplxVecIt

Definition at line 108 of file Dvbt1InterpolatorComponent.h.

7.7.2.6 typedef std::vector<float> iris::phy::Dvbt1InterpolatorComponent::FloatVec

A vector of float.

Definition at line 111 of file Dvbt1InterpolatorComponent.h.

7.7.2.7 typedef FloatVec::iterator iris::phy::Dvbt1InterpolatorComponent::FloatVecIt

An iterator for a vector of float.

Definition at line 114 of file Dvbt1InterpolatorComponent.h.

7.7.2.8 typedef std::vector<int> iris::phy::Dvbt1InterpolatorComponent::IntVec

A vector of integers.

Definition at line 117 of file Dvbt1InterpolatorComponent.h.

7.7.2.9 typedef IntVec::iterator iris::phy::Dvbt1InterpolatorComponent::IntVecIt

An iterator for a vector of typedef.

Definition at line 120 of file Dvbt1InterpolatorComponent.h.

7.7.3 Constructor & Destructor Documentation

7.7.3.1 iris::phy::Dvbt1InterpolatorComponent::Dvbt1InterpolatorComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1InterpolatorComponent.cpp.

References debug_x, inSampleRate_x, outSampleRate_x, and responseFile_x.

```
  : PhyComponent(name,                               // component name
                "dvbt1interpolator",                 // component type
                "A DVB-T1 OFDM interpolator component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  registerParameter(
    "insamplerate", "Input sampling rate (use 0 for 9142857)",
    "0.0", true, inSampleRate_x, Interval<double>(0.0,15000000.0));

  registerParameter(
75     "outsamplerate", "Output sampling rate (use 0 for 9142857)",
76     "0.0", true, outSampleRate_x, Interval<double>(0.0,15000000.0));
77
78   registerParameter(
79     "responsefile", "Text file with the interpolating impulse response",
80     "", true, responseFile_x);
81 }
```

7.7.3.2 iris::phy::Dvbt1InterpolatorComponent::∼Dvbt1InterpolatorComponent ( )

Default destructor.

Just calls destroy().

Definition at line 86 of file Dvbt1InterpolatorComponent.cpp.

References destroy().

```
87 {
88   destroy();
89 }
```

7.7.4 Member Function Documentation

7.7.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1InterpolatorComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 161 of file Dvbt1InterpolatorComponent.h.

```
161 { return &arr[0]; }
```

7.7.4.2 double ∗ iris::phy::Dvbt1InterpolatorComponent::blackman_sinc (int ∗ *n_order,* double *T,* double *dt,* int *order*) [private]

Calculate a Blackman-windowed sinc.

Parameters

n_order	order of the calculated window
T	time extension of the window
dt	sampling time
order	preferred order of the window

Returns

array containing the window taps, please remember to free when this is not needed anymore

Definition at line 272 of file Dvbt1InterpolatorComponent.cpp.

References sinc().

Referenced by setup().

```
273 {
274     int n0 = (int) floor(T / dt);
275     int i;
276     double *h_order = NULL, w = 0.0;
277     double a0 = 7938.0 / 18608.0, a1 = 9240.0 / 18608.0, a2 = 1430.0 / 18608.0;
278     double accum = 0.0;
279     *n_order = (order + 1) * n0;
280     h_order = (double *) calloc(*n_order, sizeof(double));
281     for (i = 0; i < *n_order; i++) {
```

```
282         w = a0 - a1 * cos(2.0 * M_PI * i / (*n_order - 1)) + a2 * cos(4.0 * M_PI * i / (*n_order - 1));
283         h_order[i] = w * sinc(M_PI * (i - *n_order / 2) * dt / T);
284         accum += h_order[i] * dt;
285     }
286     /*for (i = 0; i < *n_order; i++)
287         h_order[i] /= accum;*/
288 
289     return h_order;
290 }
```

7.7.4.3 void iris::phy::Dvbt1InterpolatorComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide complex values.

Definition at line 104 of file Dvbt1InterpolatorComponent.cpp.

```
107 {
108   outputTypes["output1"] = TypeInfo< Cplx >::identifier;
109 }
```

7.7.4.4 void iris::phy::Dvbt1InterpolatorComponent::destroy ( ) [private]

Destroy the component.

Definition at line 374 of file Dvbt1InterpolatorComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1InterpolatorComponent().

```
375 {
376 }
```

7.7.4.5 template<typename T , size_t N> static T∗ iris::phy::Dvbt1InterpolatorComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 163 of file Dvbt1InterpolatorComponent.h.

```
163 { return &arr[0]+N; }
```

7.7.4.6 int iris::phy::Dvbt1InterpolatorComponent::find_rational_approximation (int ∗ *num*, int ∗ *den*, double *x*, int *N*) [private]

find a rational approximation of a real value

Parameters

x	Input value to be approximated as ratio of integers
N	maximum value for the integer at the denominator
num	the integer at the numerator
den	the integer at the denominator

Returns

0 for no errors

Definition at line 202 of file Dvbt1InterpolatorComponent.cpp.

Referenced by time_buffer_size().

```
{
    int a = 0, b = 1;
    int c = 1, d = 0;
    while (b <= N && d <= N) {
        double mediant = (double) (a + c) / (double) (b + d);
        if (x == mediant) {
            if (b + d <= N) {
                *num = a + c;
                *den = b + d;
                return 0;
            } else if (d > b) {
                *num = c;
                *den = d;
                return 0;
            } else {
                *num = a;
                *den = b;
                return 0;
            }
        } else if (x > mediant) {
            a += c;
            b += d;
        } else {
            c += a;
            d += b;
        }
    }

    if (b > N) {
        *num = c;
        *den = d;
    } else {
        *num = a;
        *den = b;
    }

    return 0;
}
```

7.7.4.7 void iris::phy::Dvbt1InterpolatorComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 114 of file Dvbt1InterpolatorComponent.cpp.

References setup().

```
{
  setup();
}
```

7.7.4.8 double iris::phy::Dvbt1InterpolatorComponent::interp_response (double * *h*, int *n*, double *dt*, double *t*) [private]

interpolate a base response

Parameters

h	array of the taps of the base impulse response
n	number of taps of the base impulse response
dt	sampling time of the base impulse response
t	time at which to interpolate the base response

Returns

the value of the base response linearly interpolated at the requested time

Definition at line 309 of file Dvbt1InterpolatorComponent.cpp.

Referenced by setup().

```
{
    if (t < 0.0)
        return 0.0;
    else if (t >= n * dt)
        return 0.0;
    else {
        int n0 = (int) floor(t / dt);
        double h0 = h[n0];
        double h1 = n0 == (n - 1) ? 0.0 : h[n0 + 1];
        return h0 + ((h1 - h0) / dt) * (t - n0 * dt);
    }
}
```

7.7.4.9 void iris::phy::Dvbt1InterpolatorComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has several significant parameters

Definition at line 186 of file Dvbt1InterpolatorComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "insamplerate" || name == "outsamplerate")
  {
    destroy();
    setup();
  }
}
```

7.7.4.10 void iris::phy::Dvbt1InterpolatorComponent::process ( ) [virtual]

Main processing method.

Definition at line 120 of file Dvbt1InterpolatorComponent.cpp.

References debug_x, inLength_, inOffset_, inReg_, T1_RESAMPLE_ORDER, tiBasepointIndex_, tiHl_, tiInsize_, and tiOutsize_.

```
{
  // request input
  DataSet< Cplx > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int numbufs = (insize + inOffset_) / tiInsize_;
  int outsize = tiOutsize_ * numbufs;

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // request output
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // copy
  for(CplxVecIt init = in->data.begin(), outit = out->data.begin(),
    inRegEff_ = inReg_.begin() + T1_RESAMPLE_ORDER + 1; init < in->data.end();
    init++)
  {
    // copy
    inRegEff_[inOffset_++] = *init;

    // do the trick
    if(inOffset_ == inLength_)
    {
      // reset
      inOffset_ = 0;

          // fractional filter
          for(int j = 0; j < tiOutsize_; j++, outit++)
          {
              // current base point
              int currbp = tiBasepointIndex_[j];

              // interpolate
              Cplx temp(0,0);
              for(int k = 0; k < T1_RESAMPLE_ORDER + 1; k++)
              {
                  temp.real(temp.real() + inRegEff_[currbp - k].real() * tiHI_[k * tiOutsize_ + j]);
                  temp.imag(temp.imag() + inRegEff_[currbp - k].imag() * tiHI_[k * tiOutsize_ + j]);
              }
              *outit = temp;
          }

          // copy last values at the beginning
          copy(inReg_.end() - (T1_RESAMPLE_ORDER + 1), inReg_.end(), inReg_.begin());
    }
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate; // not sure about this: it should change!

  // release input and output
  releaseOutputDataSet("output1", out);
  releaseInputDataSet("input1", in);
}
```

7.7.4.11 void iris::phy::Dvbt1InterpolatorComponent::registerPorts ( ) [virtual]

Register the mapper ports with the IRIS system.

This component has one input that accept complex float values and one output that provides complex float values.

Definition at line 95 of file Dvbt1InterpolatorComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< Cplx >::identifier);
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.7.4.12 void iris::phy::Dvbt1InterpolatorComponent::setup ( ) [private]

Set up all our index vectors and containers.

Definition at line 324 of file Dvbt1InterpolatorComponent.cpp.

References blackman_sinc(), inLength_, inOffset_, inReg_, inSampleRate_x, interp_response(), outSampleRate_x, responseFile_x, T1_RESAMPLE_ORDER, tiBasepointIndex_, tiHI_, tiInsize_, time_buffer_size(), and tiOutsize_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // calculate factors
  if(inSampleRate_x == 0)
    inSampleRate_x = 64.0e6/7.0;
  if(outSampleRate_x == 0)
    outSampleRate_x = 64.0e6/7.0;
  time_buffer_size(0);

  // clear
  inOffset_ = 0;
  inLength_ = tiInsize_;
  inReg_.resize(inLength_ + T1_RESAMPLE_ORDER + 1);

  // interpolator basepoint
  tiBasepointIndex_.resize(tiOutsize_);
    for(int i = 0; i < tiOutsize_; i++)
        tiBasepointIndex_[i] = (int) floor(inSampleRate_x * ((double) i /
          outSampleRate_x));

  // interpolator response
  tiHI_.resize(tiOutsize_ * (T1_RESAMPLE_ORDER + 1));
    double dtbase = (1 / inSampleRate_x) / 100.0;
  int nbase = 0;
```

```
    double *hbase = blackman_sinc(&nbase, 1 / inSampleRate_x, dtbase,
      T1_RESAMPLE_ORDER);
    for(int i = 0; i < T1_RESAMPLE_ORDER + 1; i++)
  {
        for(int j = 0; j < tiOutsize_; j++)
    {
            tiHI_[i * tiOutsize_ + j] =
        (float) interp_response(hbase, nbase, dtbase,
        ((double) j / outSampleRate_x) -
        ((double) (tiBasepointIndex_[j] - i) / inSampleRate_x));
    }
    }

    // dump to file
    if(!responseFile_x.empty())
  {
        FILE *ffp = fopen(responseFile_x.c_str(), "wt");
        for(int i = 0; i < nbase; i++)
            fprintf(ffp, "%.10f\n", hbase[i]);
        fclose(ffp);
    }
  free(hbase);

}
```

7.7.4.13 double iris::phy::Dvbt1InterpolatorComponent::sinc (double *x*) [private]

sin(x)/x function

Parameters

x	Input value

Returns

The sinc of the input

Definition at line 296 of file Dvbt1InterpolatorComponent.cpp.

Referenced by blackman_sinc().

```
{
    return x == 0.0 ? 1.0 : (sin(x) / x);
}
```

7.7.4.14 int iris::phy::Dvbt1InterpolatorComponent::time_buffer_size (int *input_samples*) [private]

size correctly the interpolation buffers

Parameters

input_samples	Size, in samples, of the input buffer

Returns

Size, in samples, of the output buffer

Definition at line 246 of file Dvbt1InterpolatorComponent.cpp.

References find_rational_approximation(), inSampleRate_x, outSampleRate_x, tiInsize_, and tiOutsize_.

Referenced by setup().

```
{
    int output_samples = 0;
    int status = 0;

    // find the best rational approximation
    tiOutsize_ = 0;
    tiInsize_ = 0;
    status = find_rational_approximation(&tiOutsize_, &tiInsize_, outSampleRate_x / inSampleRate_x, 2000);
    if (status)
        LOG(LERROR) << "Could not find a rational approximation for " << outSampleRate_x << "/" << inSampleRate_x << "=" << outSampleRate_x / inSampleRate_x;

    LOG(LINFO) << "Original sampling rate: " << inSampleRate_x << " sps";
    LOG(LINFO) << "Effective sampling rate (x" << tiOutsize_ << "/" << tiInsize_ << "): " << inSampleRate_x * (double) tiOutsize_ / (double) tiInsize_ << " sps";

    return output_samples;
}
```

7.7.5 Member Data Documentation

7.7.5.1 bool iris::phy::Dvbt1InterpolatorComponent::debug_x [private]

Debug flag (default = false)

Definition at line 134 of file Dvbt1InterpolatorComponent.h.

Referenced by Dvbt1InterpolatorComponent(), and process().

7.7.5.2 int iris::phy::Dvbt1InterpolatorComponent::inLength_ [private]

Definition at line 149 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), and setup().

7.7.5.3 int iris::phy::Dvbt1InterpolatorComponent::inOffset_ [private]

Definition at line 147 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), and setup().

7.7.5.4 CplxVec iris::phy::Dvbt1InterpolatorComponent::inReg_ [private]

Definition at line 148 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), and setup().

7.7.5.5 double iris::phy::Dvbt1InterpolatorComponent::inSampleRate_x [private]

Input sampling rate (default = 0)

Definition at line 135 of file Dvbt1InterpolatorComponent.h.

Referenced by Dvbt1InterpolatorComponent(), setup(), and time_buffer_size().

7.7.5.6 double iris::phy::Dvbt1InterpolatorComponent::outSampleRate_x [private]

Output sampling rate (default = 0)

Definition at line 136 of file Dvbt1InterpolatorComponent.h.

Referenced by Dvbt1InterpolatorComponent(), setup(), and time_buffer_size().

7.7.5.7 std::string iris::phy::Dvbt1InterpolatorComponent::responseFile_x [private]

Text file with impulse response (default = none)

Definition at line 137 of file Dvbt1InterpolatorComponent.h.

Referenced by Dvbt1InterpolatorComponent(), and setup().

7.7.5.8 double iris::phy::Dvbt1InterpolatorComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 143 of file Dvbt1InterpolatorComponent.h.

7.7.5.9 IntVec iris::phy::Dvbt1InterpolatorComponent::tiBasepointIndex_ [private]

Definition at line 150 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), and setup().

7.7.5.10 FloatVec iris::phy::Dvbt1InterpolatorComponent::tiHl_ [private]

Definition at line 151 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), and setup().

7.7.5.11 int iris::phy::Dvbt1InterpolatorComponent::tiInsize_ [private]

Definition at line 145 of file Dvbt1InterpolatorComponent.h.

Referenced by process(), setup(), and time_buffer_size().

7.7.5.12 double iris::phy::Dvbt1InterpolatorComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 142 of file Dvbt1InterpolatorComponent.h.

7.7.5.13 int iris::phy::Dvbt1InterpolatorComponent::tiOutsize_ [private]

Definition at line 146 of file Dvbt1InterpolatorComponent.h.

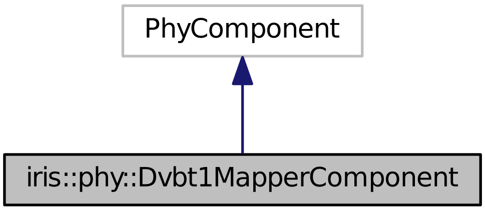
Referenced by process(), setup(), and time_buffer_size().

7.8 iris::phy::Dvbt1MapperComponent Class Reference

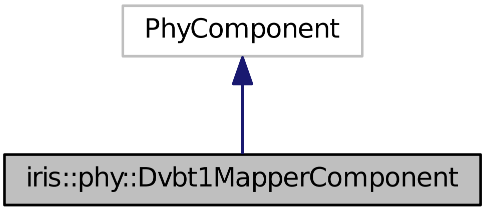
A DVB-T1 mapper component.

`#include <Dvbt1MapperComponent.h>`

Inheritance diagram for iris::phy::Dvbt1MapperComponent:



Collaboration diagram for iris::phy::Dvbt1MapperComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.
- typedef std::complex< float > Cplx

  A complex type.
- typedef std::vector< Cplx > CplxVec

  A vector of complex.
- typedef CplxVec::iterator CplxVecIt

Public Member Functions

- Dvbt1MapperComponent (std::string name)

  Default constructor.
- ∼Dvbt1MapperComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

Calculate the output port types for the IRIS system.

- virtual void registerPorts ()

  Register the mapper ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all our constellations.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int qamMapping_x

  QAM constellation mapping (default = 16)
- int hyerarchyMode_x

  Hyerarchical mode (default = 0)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- CplxVec constel_

  actual constellation

7.8.1 Detailed Description

A DVB-T1 mapper component.

Dvbt1MapperComponent is the eighth block composing the DVB-T transmission chain. The mapper uses the QAM constellations mandated in the standard, to transform the data symbols into complex numbers that can be eventually delivered over I&Q analog waveforms. The constellations are Gray-encoded, that is, adjacent points in the complex plane only differ in one bit among their represented symbols (indeed, this is only partially true, as there can also be a difference of more bits, but at larger distances).

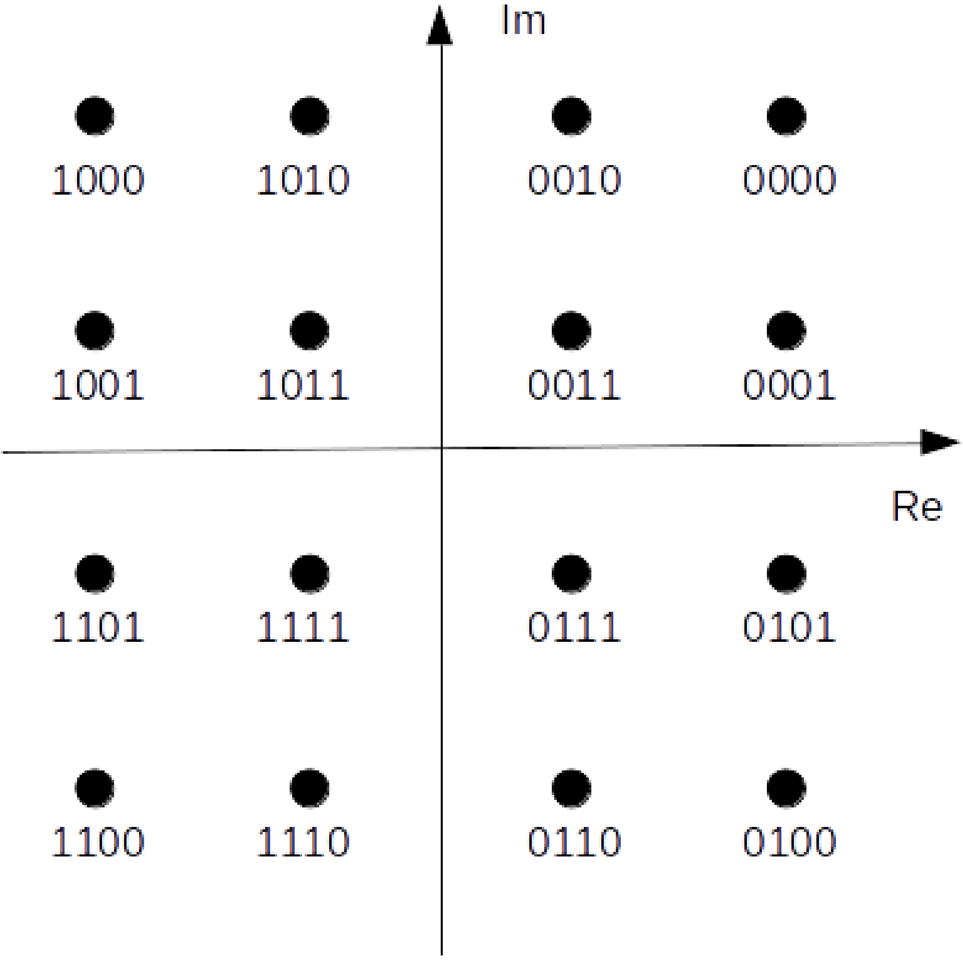


Figure 7.6: DVB-T 16-QAM constellation.

The constellation points are statically written in the source files.

This blocks accepts in input elements in uint8_t (ν-bit symbols) and generates in output complex values (complex float).

There are three parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *qammapping*: by default set to "16", this is used to select one of the three possible QAM mappings. The admitted values are "4", "16", "64".
- *hyerarchymode*: by default set to "0", which means "not hyerarchical". Hierarchical modes are used to transmit two different transport streams, one with a high priority (HP) information and another one with a low priority (LP) information. The admitted values are "0, "1", "2", "4". NOTE: hyerarchical modes are not implemented in the current release of this modulator.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

Definition at line 83 of file Dvbt1MapperComponent.h.

7.8.2 Member Typedef Documentation

7.8.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1MapperComponent::ByteVec

A vector of bytes.

Definition at line 89 of file Dvbt1MapperComponent.h.

7.8.2.2 typedef ByteVec::iterator iris::phy::Dvbt1MapperComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 92 of file Dvbt1MapperComponent.h.

7.8.2.3 typedef std::complex<float> iris::phy::Dvbt1MapperComponent::Cplx

A complex type.

Definition at line 95 of file Dvbt1MapperComponent.h.

7.8.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1MapperComponent::CplxVec

A vector of complex.

Definition at line 98 of file Dvbt1MapperComponent.h.

7.8.2.5 typedef CplxVec::iterator iris::phy::Dvbt1MapperComponent::CplxVecIt

Definition at line 101 of file Dvbt1MapperComponent.h.

7.8.3 Constructor & Destructor Documentation

7.8.3.1 iris::phy::Dvbt1MapperComponent::Dvbt1MapperComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1MapperComponent.cpp.

References begin(), debug_x, end(), hyerarchyMode_x, and qamMapping_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1mapper",                    // component type
                "A DVB-T1 mapper component",      // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  int qamarr[] = {4,16,64};
  registerParameter(
    "qammapping", "QAM constellation mapping",
    "16", true, qamMapping_x, list<int>(begin(qamarr),end(qamarr)));

  int harr[] = {0,1,2,4};
  registerParameter(
    "hyerarchymode", "Hyerarchical mode (0 = NH)",
    "0", true, hyerarchyMode_x, list<int>(begin(harr),end(harr)));
}
```

7.8.3.2 iris::phy::Dvbt1MapperComponent::∼Dvbt1MapperComponent ( )

Default destructor.

Just calls destroy().

Definition at line 84 of file Dvbt1MapperComponent.cpp.

References destroy().

```
85 {
86   destroy();
87 }
```

7.8.4 Member Function Documentation

7.8.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1MapperComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 129 of file Dvbt1MapperComponent.h.

Referenced by Dvbt1MapperComponent().

```
129 { return &arr[0]; }
```

7.8.4.2 void iris::phy::Dvbt1MapperComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide complex values.

Definition at line 102 of file Dvbt1MapperComponent.cpp.

```
105 {
106   outputTypes["output1"] = TypeInfo< Cplx >::identifier;
107 }
```

7.8.4.3 void iris::phy::Dvbt1MapperComponent::destroy ( ) [private]

Destroy the component.

Definition at line 284 of file Dvbt1MapperComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1MapperComponent().

```
285 {
286 }
```

7.8.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1MapperComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 131 of file Dvbt1MapperComponent.h.

Referenced by Dvbt1MapperComponent().

```
131 { return &arr[0]+N; }
```

7.8.4.5 void iris::phy::Dvbt1MapperComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 112 of file Dvbt1MapperComponent.cpp.

References setup().

```
{
  setup();
}
```

7.8.4.6 void iris::phy::Dvbt1MapperComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has two significant parameters

Definition at line 155 of file Dvbt1MapperComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "qammapping" || name == "hyerarchymode")
  {
    destroy();
    setup();
  }
}
```

7.8.4.7 void iris::phy::Dvbt1MapperComponent::process ( ) [virtual]

Main processing method.

Definition at line 118 of file Dvbt1MapperComponent.cpp.

References constel_, and debug_x.

```
{
  // request input
  DataSet< uint8_t > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = insize;

  // request output
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // assign
  CplxVecIt outit = out->data.begin();
  for(ByteVecIt init = in->data.begin(); init < in->data.end(); init++, outit++)
  {
    *outit = constel_[*init];
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.8.4.8 void iris::phy::Dvbt1MapperComponent::registerPorts ( ) [virtual]

Register the mapper ports with the IRIS system.

This component has one input that accept symbols (some bits per byte) and one output that provides complex symbols (in floats).

Definition at line 93 of file Dvbt1MapperComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.8.4.9 void iris::phy::Dvbt1MapperComponent::setup ( ) [private]

Set up all our constellations.

Definition at line 165 of file Dvbt1MapperComponent.cpp.

References constel_, hyerarchyMode_x, and qamMapping_x.

Referenced by initialize(), and parameterHasChanged().

```
{
  // nonuniformity value
    float alpha = hyerarchyMode_x == 0 ? 1 : (float) ceil((double) (1 <<
  hyerarchyMode_x) / 2.0);

    // constellation array
  switch(qamMapping_x)
  {
    case 4:
      constel_.push_back(Cplx(1, 1));
      constel_.push_back(Cplx(1, -1));
      constel_.push_back(Cplx(-1, 1));
      constel_.push_back(Cplx(-1, -1));
      break;
    case 16:
      constel_.push_back(Cplx(3, 3));
      constel_.push_back(Cplx(3, 1));
      constel_.push_back(Cplx(1, 3));
      constel_.push_back(Cplx(1, 1));
      constel_.push_back(Cplx(3, -3));
      constel_.push_back(Cplx(3, -1));
      constel_.push_back(Cplx(1, -3));
      constel_.push_back(Cplx(1, -1));
      constel_.push_back(Cplx(-3, 3));
      constel_.push_back(Cplx(-3, 1));
      constel_.push_back(Cplx(-1, 3));
      constel_.push_back(Cplx(-1, 1));
      constel_.push_back(Cplx(-3, -3));
      constel_.push_back(Cplx(-3, -1));
      constel_.push_back(Cplx(-1, -3));
      constel_.push_back(Cplx(-1, -1));
      break;
    case 64:
      constel_.push_back(Cplx(7, 7));
      constel_.push_back(Cplx(7, 5));
      constel_.push_back(Cplx(5, 7));
      constel_.push_back(Cplx(5, 5));
      constel_.push_back(Cplx(7, 1));
      constel_.push_back(Cplx(7, 3));
      constel_.push_back(Cplx(5, 1));
      constel_.push_back(Cplx(5, 3));
      constel_.push_back(Cplx(1, 7));
      constel_.push_back(Cplx(1, 5));
      constel_.push_back(Cplx(3, 7));
      constel_.push_back(Cplx(3, 5));
      constel_.push_back(Cplx(1, 1));
      constel_.push_back(Cplx(1, 3));
      constel_.push_back(Cplx(3, 1));
      constel_.push_back(Cplx(3, 3));
      constel_.push_back(Cplx(7, -7));
      constel_.push_back(Cplx(7, -5));
      constel_.push_back(Cplx(5, -7));
      constel_.push_back(Cplx(5, -5));
      constel_.push_back(Cplx(7, -1));
      constel_.push_back(Cplx(7, -3));
      constel_.push_back(Cplx(5, -1));
      constel_.push_back(Cplx(5, -3));
      constel_.push_back(Cplx(1, -7));
      constel_.push_back(Cplx(1, -5));
      constel_.push_back(Cplx(3, -7));
      constel_.push_back(Cplx(3, -5));
      constel_.push_back(Cplx(1, -1));
      constel_.push_back(Cplx(1, -3));
      constel_.push_back(Cplx(3, -1));
      constel_.push_back(Cplx(3, -3));
      constel_.push_back(Cplx(-7, 7));
      constel_.push_back(Cplx(-7, 5));
      constel_.push_back(Cplx(-5, 7));
      constel_.push_back(Cplx(-5, 5));
      constel_.push_back(Cplx(-7, 1));
      constel_.push_back(Cplx(-7, 3));
      constel_.push_back(Cplx(-5, 1));
      constel_.push_back(Cplx(-5, 3));
      constel_.push_back(Cplx(-1, 7));
      constel_.push_back(Cplx(-1, 5));
      constel_.push_back(Cplx(-3, 7));
      constel_.push_back(Cplx(-3, 5));
      constel_.push_back(Cplx(-1, 1));
      constel_.push_back(Cplx(-1, 3));
      constel_.push_back(Cplx(-3, 1));
      constel_.push_back(Cplx(-3, 3));
      constel_.push_back(Cplx(-7, -7));
      constel_.push_back(Cplx(-7, -5));
      constel_.push_back(Cplx(-5, -7));
      constel_.push_back(Cplx(-5, -5));
      constel_.push_back(Cplx(-7, -1));
      constel_.push_back(Cplx(-7, -3));
      constel_.push_back(Cplx(-5, -1));
      constel_.push_back(Cplx(-5, -3));
      constel_.push_back(Cplx(-1, -7));
      constel_.push_back(Cplx(-1, -5));
      constel_.push_back(Cplx(-3, -7));
      constel_.push_back(Cplx(-3, -5));
      constel_.push_back(Cplx(-1, -1));
      constel_.push_back(Cplx(-1, -3));
      constel_.push_back(Cplx(-3, -1));
      constel_.push_back(Cplx(-3, -3));
      break;
  }

  // add alpha and find energy
  float energy = 0;
  for(int m = 0; m < constel_.size(); m++)
  {
    constel_[m].real(constel_[m].real() + (alpha - 1) * (constel_[m].real() >= 0 ? 1 : -1));
    constel_[m].imag(constel_[m].imag() + (alpha - 1) * (constel_[m].imag() >= 0 ? 1 : -1));
    energy += constel_[m].real() * constel_[m].real() + constel_[m].imag() * constel_[m].imag();
  }
    energy = sqrtf(energy / constel_.size());

  // normalize to have unit energy
  for(int m = 0; m < constel_.size(); m++)
  {
        constel_[m].real(constel_[m].real() / energy);
        constel_[m].imag(constel_[m].imag() / energy);
    }
}
```

7.8.5 Member Data Documentation

7.8.5.1 CplxVec iris::phy::Dvbt1MapperComponent::constel_ [private]

actual constellation

Definition at line 125 of file Dvbt1MapperComponent.h.

Referenced by process(), and setup().

7.8.5.2 bool iris::phy::Dvbt1MapperComponent::debug_x [private]

Debug flag (default = false)

Definition at line 115 of file Dvbt1MapperComponent.h.

Referenced by Dvbt1MapperComponent(), and process().

7.8.5.3 int iris::phy::Dvbt1MapperComponent::hyerarchyMode_x [private]

Hyerarchical mode (default = 0)

Definition at line 117 of file Dvbt1MapperComponent.h.

Referenced by Dvbt1MapperComponent(), and setup().

7.8.5.4 int iris::phy::Dvbt1MapperComponent::qamMapping_x [private]

QAM constellation mapping (default = 16)

Definition at line 116 of file Dvbt1MapperComponent.h.

Referenced by Dvbt1MapperComponent(), and setup().

7.8.5.5 double iris::phy::Dvbt1MapperComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 123 of file Dvbt1MapperComponent.h.

7.8.5.6 double iris::phy::Dvbt1MapperComponent::timeStamp_ [private]

Timestamp of current frame.

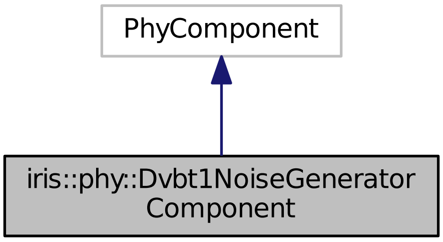
Definition at line 122 of file Dvbt1MapperComponent.h.

7.9 iris::phy::Dvbt1NoiseGeneratorComponent Class Reference

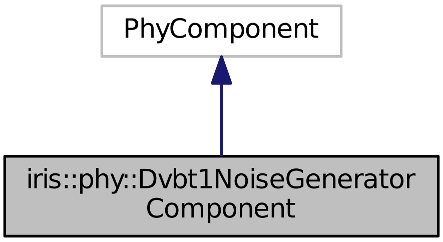
A DVB-T1 noise generator.

`#include <Dvbt1NoiseGeneratorComponent.h>`

Inheritance diagram for iris::phy::Dvbt1NoiseGeneratorComponent:



Collaboration diagram for iris::phy::Dvbt1NoiseGeneratorComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec
- typedef ByteVec::iterator ByteVecIt
- typedef std::complex< float > Cplx
- typedef std::vector< Cplx > CplxVec
- typedef CplxVec::iterator CplxVecIt
- typedef std::vector< float > FloatVec
- typedef FloatVec::iterator FloatVecIt
- typedef std::vector< int > IntVec
- typedef IntVec::iterator IntVecIt
- typedef
  boost::normal_distribution
  < double > NormalDistribution
- typedef boost::mt19937 RandomGenerator
- typedef
  boost::variate_generator
  < RandomGenerator,
  NormalDistribution > GaussianGenerator

Public Member Functions

- Dvbt1NoiseGeneratorComponent (std::string name)
- ∼Dvbt1NoiseGeneratorComponent ()
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)
- virtual void registerPorts ()
- virtual void initialize ()
- virtual void process ()
- virtual void parameterHasChanged (std::string name)

Private Member Functions

- void setup ()

  Set up all our index vectors and containers.
- void destroy ()

Static Private Member Functions

- template<typename T , size_t N>
static T ∗ begin (T(&arr)[N])
- template<typename T , size_t N>
static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double variance_x

  Noise variance (default = 1)
- int blockSize_x

  Size of blocks to generate.
- double frequency_x

  Size of blocks to generate.
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- double theta0_
- GaussianGenerator ∗ gen

7.9.1 Detailed Description

A DVB-T1 noise generator.

Definition at line 50 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2 Member Typedef Documentation

7.9.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1NoiseGeneratorComponent::ByteVec

Definition at line 55 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.2 typedef ByteVec::iterator iris::phy::Dvbt1NoiseGeneratorComponent::ByteVecIt

Definition at line 56 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.3 typedef std::complex<float> iris::phy::Dvbt1NoiseGeneratorComponent::Cplx

Definition at line 57 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.4 typedef std::vector<Cplx> iris::phy::Dvbt1NoiseGeneratorComponent::CplxVec

Definition at line 58 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.5 typedef CplxVec::iterator iris::phy::Dvbt1NoiseGeneratorComponent::CplxVecIt

Definition at line 59 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.6 typedef std::vector<float> iris::phy::Dvbt1NoiseGeneratorComponent::FloatVec

Definition at line 60 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.7 typedef FloatVec::iterator iris::phy::Dvbt1NoiseGeneratorComponent::FloatVecIt

Definition at line 61 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.8 typedef boost::variate_generator<RandomGenerator,NormalDistribution> iris::phy::Dvbt1NoiseGeneratorComponent::GaussianGenerator

Definition at line 66 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.9 typedef std::vector<int> iris::phy::Dvbt1NoiseGeneratorComponent::IntVec

Definition at line 62 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.10 typedef IntVec::iterator iris::phy::Dvbt1NoiseGeneratorComponent::IntVecIt

Definition at line 63 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.11 typedef boost::normal_distribution<double> iris::phy::Dvbt1NoiseGeneratorComponent::NormalDistribution

Definition at line 64 of file Dvbt1NoiseGeneratorComponent.h.

7.9.2.12 typedef boost::mt19937 iris::phy::Dvbt1NoiseGeneratorComponent::RandomGenerator

Definition at line 65 of file Dvbt1NoiseGeneratorComponent.h.

7.9.3 Constructor & Destructor Documentation

7.9.3.1 iris::phy::Dvbt1NoiseGeneratorComponent::Dvbt1NoiseGeneratorComponent (std::string *name*)

Definition at line 54 of file Dvbt1NoiseGeneratorComponent.cpp.

References blockSize_x, debug_x, frequency_x, and variance_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1noisegenerator",                 // component type
                "A DVB-T1 noise generator component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,theta0_(0)
{
  registerParameter(
    "variance", "Noise variance (default = 1)",
    "1", true, variance_x, Interval<double>(0.0,1000000.0));

  registerParameter(
    "frequency", "Cosine frequency",
    "3e6", true, frequency_x, Interval<double>(0.0,100000000.0));

  registerParameter("blocksize",
                    "Size of generated blocks",
                    "1024",
                    true,
                    blockSize_x,
77                    Interval<int>(1, 128*1024*1024));
78
79   registerParameter(
80     "debug", "Whether to output debug data",
81     "false", true, debug_x);
82 }
```

7.9.3.2 iris::phy::Dvbt1NoiseGeneratorComponent::∼Dvbt1NoiseGeneratorComponent ( )

Definition at line 84 of file Dvbt1NoiseGeneratorComponent.cpp.

References destroy().

```
85 {
86   destroy();
87 }
```

7.9.4 Member Function Documentation

7.9.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1NoiseGeneratorComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 95 of file Dvbt1NoiseGeneratorComponent.h.

```
95 { return &arr[0]; }
```

7.9.4.2 void iris::phy::Dvbt1NoiseGeneratorComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Definition at line 94 of file Dvbt1NoiseGeneratorComponent.cpp.

```
97 {
98   outputTypes["output1"] = TypeInfo< Cplx >::identifier;
99 }
```

7.9.4.3 void iris::phy::Dvbt1NoiseGeneratorComponent::destroy ( ) [private]

Definition at line 151 of file Dvbt1NoiseGeneratorComponent.cpp.

References gen.

Referenced by parameterHasChanged(), and ∼Dvbt1NoiseGeneratorComponent().

```
152 {
153   delete gen;
154 }
```

7.9.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1NoiseGeneratorComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 97 of file Dvbt1NoiseGeneratorComponent.h.

```
97 { return &arr[0]+N; }
```

7.9.4.5 void iris::phy::Dvbt1NoiseGeneratorComponent::initialize ( ) [virtual]

Definition at line 101 of file Dvbt1NoiseGeneratorComponent.cpp.

References setup().

```
{
  setup();
}
```

7.9.4.6 void iris::phy::Dvbt1NoiseGeneratorComponent::parameterHasChanged (std::string *name*) [virtual]

Definition at line 133 of file Dvbt1NoiseGeneratorComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "variance")
  {
    destroy();
    setup();
  }
}
```

7.9.4.7 void iris::phy::Dvbt1NoiseGeneratorComponent::process ( ) [virtual]

Definition at line 106 of file Dvbt1NoiseGeneratorComponent.cpp.

References blockSize_x, debug_x, frequency_x, theta0_, and variance_x.

```
{
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, blockSize_x);

  if(debug_x)
    LOG(LINFO) << "out: " << blockSize_x;

  float dtheta = 2*M_PI*frequency_x/12.5e6;
  float A = sqrt(variance_x * 2);
  int n = 0;
  for(CplxVecIt outit = out->data.begin(); outit < out->data.end(); outit++)
  {
    /*outit->real((*gen)());
    outit->imag((*gen)());*/
    outit->real(A*cos(theta0_ + dtheta*(n++)));
    outit->imag(0);
  }
  theta0_ += dtheta*n;

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = 0;
  out->sampleRate = 0;

  releaseOutputDataSet("output1", out);
}
```

7.9.4.8 void iris::phy::Dvbt1NoiseGeneratorComponent::registerPorts ( ) [virtual]

Definition at line 89 of file Dvbt1NoiseGeneratorComponent.cpp.

```
{
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.9.4.9 void iris::phy::Dvbt1NoiseGeneratorComponent::setup ( ) [private]

Set up all our index vectors and containers.

Definition at line 143 of file Dvbt1NoiseGeneratorComponent.cpp.

References gen, theta0_, and variance_x.

Referenced by initialize(), and parameterHasChanged().

```
{
  RandomGenerator  eng;
  NormalDistribution dist(0, variance_x / 2);
  gen = new GaussianGenerator(eng,dist);
  theta0_ = 0;
}
```

7.9.5 Member Data Documentation

7.9.5.1 int iris::phy::Dvbt1NoiseGeneratorComponent::blockSize_x [private]

Size of blocks to generate.

Definition at line 82 of file Dvbt1NoiseGeneratorComponent.h.

Referenced by Dvbt1NoiseGeneratorComponent(), and process().

7.9.5.2 bool iris::phy::Dvbt1NoiseGeneratorComponent::debug_x [private]

Debug flag (default = false)

Definition at line 80 of file Dvbt1NoiseGeneratorComponent.h.

Referenced by Dvbt1NoiseGeneratorComponent(), and process().

7.9.5.3 double iris::phy::Dvbt1NoiseGeneratorComponent::frequency_x [private]

Size of blocks to generate.

Definition at line 83 of file Dvbt1NoiseGeneratorComponent.h.

Referenced by Dvbt1NoiseGeneratorComponent(), and process().

7.9.5.4 GaussianGenerator* iris::phy::Dvbt1NoiseGeneratorComponent::gen [private]

Definition at line 92 of file Dvbt1NoiseGeneratorComponent.h.

Referenced by destroy(), and setup().

7.9.5.5 double iris::phy::Dvbt1NoiseGeneratorComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 89 of file Dvbt1NoiseGeneratorComponent.h.

7.9.5.6 double iris::phy::Dvbt1NoiseGeneratorComponent::theta0_ [private]

Definition at line 90 of file Dvbt1NoiseGeneratorComponent.h.

Referenced by process(), and setup().

7.9.5.7 double iris::phy::Dvbt1NoiseGeneratorComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 88 of file Dvbt1NoiseGeneratorComponent.h.

7.9.5.8 double iris::phy::Dvbt1NoiseGeneratorComponent::variance_x [private]

Noise variance (default = 1)

Definition at line 81 of file Dvbt1NoiseGeneratorComponent.h.

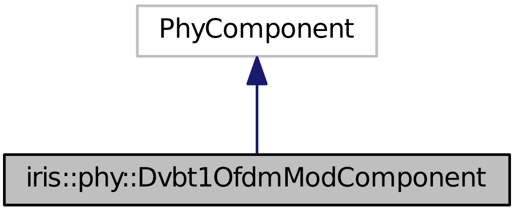
Referenced by Dvbt1NoiseGeneratorComponent(), process(), and setup().

7.10 iris::phy::Dvbt1OfdmModComponent Class Reference

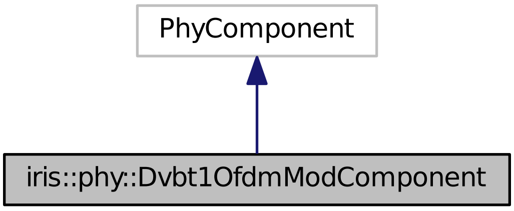
A DVB-T1 OFDM modulator component.

`#include <Dvbt1OfdmModComponent.h>`

Inheritance diagram for iris::phy::Dvbt1OfdmModComponent:



Collaboration diagram for iris::phy::Dvbt1OfdmModComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.
- typedef std::complex< float > Cplx

  A complex type.
- typedef std::vector< Cplx > CplxVec

  A vector of complex.
- typedef CplxVec::iterator CplxVecIt
- typedef std::vector< float > FloatVec

  A vector of float.
- typedef FloatVec::iterator FloatVecIt

  An iterator for a vector of float.
- typedef std::vector< int > IntVec

  A vector of integers.
- typedef IntVec::iterator IntVecIt

  An iterator for a vector of typedef.

Public Member Functions

- Dvbt1OfdmModComponent (std::string name)

  Default constructor.
- ∼Dvbt1OfdmModComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the mapper ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all needed constants.
- void destroy ()

  Destroy the component.
- void powerProcedure_ ()

  Separate thread for power loading.
- double sinc (double x)

  sin(x)/x function
- double frequency_response_modulus (double ∗h, int n, double dt, double f)

  Calculate the frequency response modulus of an impulse response.
- double ∗ blackman_sinc (int ∗n_order, double T, double dt, int order)

  Calculate a Blackman-windowed sinc.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int ofdmMode_x

  OFDM mode (default = 2048)
- int deltaMode_x

  Cyclic prefix ratio (default = 32)
- float outPower_x

  Output power indicator (default = 10)
- double dacSampleRate_x

  Sampling rate used by the DAC.
- std::string powerFile_x

  Text file with power loading (default = none)
- double powerInterval_x

  Power update interval in seconds (default = 0)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int nFft_
- int nDelta_
- int nBlock_
- int inOffset_
- CplxVec inReg_
- CplxVec fftReg_
- int nMax_
- int kMax_
- int tpsNum_
- int nBit_
- float multFactor_
- FloatVec _precorrFactor_
- FloatVecIt precorrFactor_
- FloatVec _ampliFactor_
- FloatVecIt ampliFactor_
- fftwf_plan fft_

  Our FFT object pointer.
- Cplx ∗ fftBins_

  Allocated using fftwf_malloc (SIMD aligned)
- boost::thread ∗ powerThread_
- bool runPower_

7.10.1 Detailed Description

A DVB-T1 OFDM modulator component.

Dvbt1OfdmModComponent is the tenth block composing the DVB-T transmission chain. The OFDM block takes the modulated QAM cells, assembled in frames together with the pilot and TPS cells, and converts them from a *virtual* frequency domain sequence to a time domain signal, which can be transmitted on a channel.

Not all the carriers are modulated, but some of them are left untouched for purposes of guard bandwidth implementation (*virtual* carriers). Due to the peculiar way frequencies are structured in the sampled frequency domain, the central part of the spectrum is left to the virtual carriers, whereas the outer portions are occupied by the active carriers.

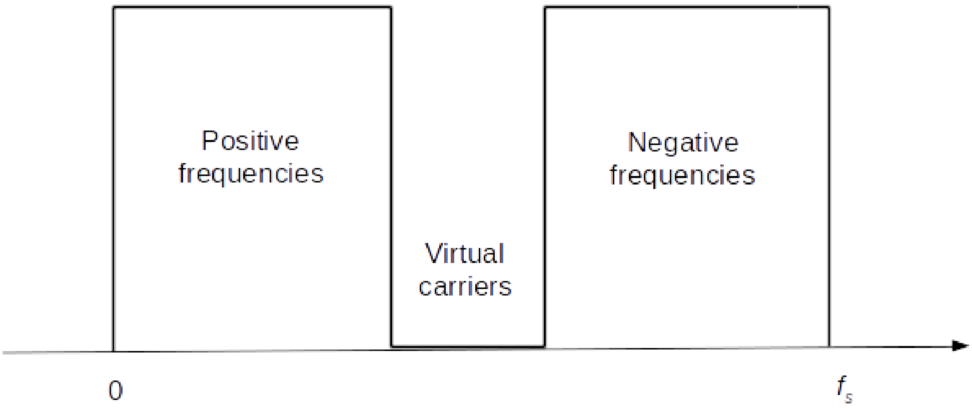


Figure 7.7: OFDM carriers arrangement.

The conversion between the frequency and time domains can be done in several ways, either by using a bank of quadrature modulators or an inverse Discrete Frequency Transform algorithm. In our case, we use an inverse Fast Fourier Transform algorithm, and the signal generated starting from the active cells Ψ_k can be written as

$$x[n] = \frac{1}{N_{\mathrm{FFT}}} \sum_{k=0}^{N_{\mathrm{FFT}}-1} \rho_k \Lambda_k \Psi_k e^{j\frac{2\pi}{N_{\mathrm{FFT}}}kn}, \quad n = -L, -(L-1), \ldots, 1, 0, 1, 2, \ldots, (N_{\mathrm{FFT}}-1),$$

where L is the cyclic prefix size, ρ_k is a frequency amplitude linear precorrection term and Λ_k is a power-loading factor: the purpose of these terms will be clarified below.

This block accepts in input complex float values and generates in output complex float values.

There are several parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *ofdmmode*: by default set to "2048", this is used to select one of the three possible OFDM modes. The admitted values are "2048", "4096", "8192", respectively for 2K, 4K (DVB-H, unused), and 8K.
- *deltamode*: by default set to "32", this is used to select one of the four possible cyclic prefix lengths. The admitted values are "32", "16", "8", and "4", which are directly derived from the denominator of the cyclic prefix fraction (1/32, 1/16, 1/8, 1/4).
- *outpower*: by default set to "10", this parameter represents the scaling factor used for rescaling the IFFT output into the wanted range. In particular, this parameter is a percentage. A percentage of 100 means that the output signal real and imaginary parts have an amplitude distribution that concentrates the values into a interval between -1 and 1 with the 99.7% of probability. Since the OFDM signal is Gaussian, this means that the $\pm 3\sigma$ interval of amplitudes falls in the span $[-1, +1]$. When the digital signal is mapped onto analog values by the USRP DAC, for example, the valid range is that enclosed in the $[-1, +1]$ interval, all other values will be clipped.

- *dacsamplerate*: by default set to "0", a placeholder for 64e6/7 Hz. This represents the sampling rate adopted by the DAC for emitting the BB analog signal. It is used internally to precorrect, linearly, with a multiplicative factor ρ_k, the amplitude of the OFDM carriers that will be distorted by the Dvbt1Interpolator block. The type of distortion is decided by the algorithm adopted internally by the interpolator block. **Please note that if you are not using the Dvbt1Interpolator block, then you need to leave this parameter at 0**.
- *powerfile*: by default empty, this is the name of a text file that can be read, at periodic intervals, to generate a powerloading configuration for the OFDM carriers. This file contains, line by line, the value of power correction, expressed in dB, for each one of the OFDM carriers. For instance, for 8K OFDM, the file is composed by 8192 lines. A value of 0 means that the power of the carrier is left untouched, a positive valiue means that there will be a power increase, a negative value will result into a power decrease. The positioning of the carrier indices starts from the first, lowest frequency carrier up to the last, highest frequency carrier.
- *powerinterval*: by default it is set to "1". This is the number of seconds among consecutive reads of the power loading file.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

Definition at line 139 of file Dvbt1OfdmModComponent.h.

7.10.2 Member Typedef Documentation

7.10.2.1 typedef std::vector<uint8_t> **iris::phy::Dvbt1OfdmModComponent::ByteVec**

A vector of bytes.

Definition at line 145 of file Dvbt1OfdmModComponent.h.

7.10.2.2 typedef ByteVec::iterator **iris::phy::Dvbt1OfdmModComponent::ByteVecIt**

An iterator for a vector of bytes.

Definition at line 148 of file Dvbt1OfdmModComponent.h.

7.10.2.3 typedef std::complex<float> **iris::phy::Dvbt1OfdmModComponent::Cplx**

A complex type.

Definition at line 151 of file Dvbt1OfdmModComponent.h.

7.10.2.4 typedef std::vector<**Cplx**> **iris::phy::Dvbt1OfdmModComponent::CplxVec**

A vector of complex.

Definition at line 154 of file Dvbt1OfdmModComponent.h.

7.10.2.5 typedef CplxVec::iterator **iris::phy::Dvbt1OfdmModComponent::CplxVecIt**

Definition at line 157 of file Dvbt1OfdmModComponent.h.

7.10.2.6 typedef std::vector<float> **iris::phy::Dvbt1OfdmModComponent::FloatVec**

A vector of float.

Definition at line 160 of file Dvbt1OfdmModComponent.h.

7.10.2.7 typedef FloatVec::iterator iris::phy::Dvbt1OfdmModComponent::FloatVecIt

An iterator for a vector of float.

Definition at line 163 of file Dvbt1OfdmModComponent.h.

7.10.2.8 typedef std::vector<int> iris::phy::Dvbt1OfdmModComponent::IntVec

A vector of integers.

Definition at line 166 of file Dvbt1OfdmModComponent.h.

7.10.2.9 typedef IntVec::iterator iris::phy::Dvbt1OfdmModComponent::IntVecIt

An iterator for a vector of typedef.

Definition at line 169 of file Dvbt1OfdmModComponent.h.

7.10.3 Constructor & Destructor Documentation

7.10.3.1 iris::phy::Dvbt1OfdmModComponent::Dvbt1OfdmModComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 59 of file Dvbt1OfdmModComponent.cpp.

References begin(), dacSampleRate_x, debug_x, deltaMode_x, end(), ofdmMode_x, outPower_x, powerFile_x, and powerInterval_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1ofdmmod",                   // component type
                "A DVB-T1 OFDM modulator component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,fft_(NULL)
    ,fftBins_(NULL)
    ,powerThread_(NULL)
    ,runPower_(false)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  int ofdmarr[] = {2048,4096,8192};
  registerParameter(
    "ofdmmode", "OFDM mode",
    "2048", true, ofdmMode_x, list<int>(begin(ofdmarr),end(ofdmarr)));

  int deltaarr[] = {32,16,8,4};
  registerParameter(
    "deltamode", "Cyclic prefix ratio",
    "32", true, deltaMode_x, list<int>(begin(deltaarr),end(deltaarr)));

  registerParameter(
    "outpower", "Output power, in percentage: note that a value of 100 will "
    "result in signal clipping only below -3 sigma and above +3 sigma",
    "10", true, outPower_x, Interval<float>(0,300));

  registerParameter(
    "dacsamplerate", "Sampling rate at the DAC (default = 0, means 64e6/7)",
    "0", true, dacSampleRate_x, Interval<double>(0,15000000));

  registerParameter(
    "powerfile", "Text file with the power loading profile (default = none)",
    "", true, powerFile_x);

  registerParameter(
    "powerinterval", "Power update interval in seconds (default = 1)",
    "0", true, powerInterval_x);
102 }
```

7.10.3.2 iris::phy::Dvbt1OfdmModComponent::∼Dvbt1OfdmModComponent ( )

Default destructor.

Just calls destroy().

Definition at line 107 of file Dvbt1OfdmModComponent.cpp.

References destroy().

```
108 {
109   destroy();
110 }
```

7.10.4 Member Function Documentation

7.10.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1OfdmModComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 225 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent().

```
225 { return &arr[0]; }
```

7.10.4.2 double ∗ iris::phy::Dvbt1OfdmModComponent::blackman_sinc (int ∗ *n_order,* double *T,* double *dt,* int *order*) [private]

Calculate a Blackman-windowed sinc.

Parameters

n_order	order of the calculated window
T	time extension of the window
dt	sampling time
order	preferred order of the window

Returns

array containing the window taps, please remember to free when this is not needed anymore

Definition at line 274 of file Dvbt1OfdmModComponent.cpp.

References sinc().

Referenced by setup().

```
275 {
276     int n0 = (int) floor(T / dt);
277     int i;
278     double *h_order = NULL, w = 0.0;
279     double a0 = 7938.0 / 18608.0, a1 = 9240.0 / 18608.0, a2 = 1430.0 / 18608.0;
280     double accum = 0.0;
281     *n_order = (order + 1) * n0;
282     h_order = (double *) calloc(*n_order, sizeof(double));
283     for (i = 0; i < *n_order; i++) {
284         w = a0 - a1 * cos(2.0 * M_PI * i / (*n_order - 1)) + a2 * cos(4.0 * M_PI * i / (*n_order - 1));
285         h_order[i] = w * sinc(M_PI * (i - *n_order / 2) * dt / T);
286         accum += h_order[i] * dt;
287     }
288     /*for (i = 0; i < *n_order; i++)
289         h_order[i] /= accum;*/
290 
291     return h_order;
292 }
```

7.10.4.3 void iris::phy::Dvbt1OfdmModComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide complex values.

Definition at line 125 of file Dvbt1OfdmModComponent.cpp.

```
128 {
129   outputTypes["output1"] = TypeInfo< Cplx >::identifier;
130 }
```

7.10.4.4 void iris::phy::Dvbt1OfdmModComponent::destroy ( ) [private]

Destroy the component.

Definition at line 446 of file Dvbt1OfdmModComponent.cpp.

References fft_, fftBins_, powerThread_, and runPower_.

Referenced by parameterHasChanged(), and ∼Dvbt1OfdmModComponent().

```
447 {
448   if(fftBins_ != NULL)
449     fftwf_free(fftBins_);
450   if(fft_ != NULL)
451     fftwf_destroy_plan(fft_);
452 
453   // stop thread
454   runPower_ = false;
455   if (powerThread_) {
456     powerThread_->join();
457     delete powerThread_;
458   }
459 }
```

7.10.4.5 template<typename T , size_t N> static T∗ iris::phy::Dvbt1OfdmModComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 227 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent().

```
227 { return &arr[0]+N; }
```

7.10.4.6 double iris::phy::Dvbt1OfdmModComponent::frequency_response_modulus (double ∗ *h*, int *n*, double *dt*, double *f*) [private]

Calculate the frequency response modulus of an impulse response.

Parameters

h	array of impulse response taps
n	number of taps
dt	sampling period of the impulse response
f	frequency at which the modulus of the frequency response is calculated

Returns

the modulus of the frequency response at the indicated frequency

Definition at line 251 of file Dvbt1OfdmModComponent.cpp.

Referenced by setup().

```
{
    double H_re = 0.0, H_im = 0.0;
    double arg = 2.0 * M_PI * f * dt;
    int i;

  // just use plain old DFT, no FFT here
    for (i = 0; i < n; i++) {
        H_re += h[i] * cos(arg * i) * dt;
        H_im += h[i] * (-sin(arg * i)) * dt;
    }

    return sqrt(H_re * H_re + H_im * H_im);
}
```

7.10.4.7 void iris::phy::Dvbt1OfdmModComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 135 of file Dvbt1OfdmModComponent.cpp.

References setup().

```
{
  setup();
}
```

7.10.4.8 void iris::phy::Dvbt1OfdmModComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has several significant parameters

Definition at line 225 of file Dvbt1OfdmModComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "deltamode" || name == "ofdmmode" ||
      name == "outpower")
  {
    destroy();
    setup();
  }
}
```

7.10.4.9 void iris::phy::Dvbt1OfdmModComponent::powerProcedure_ ( ) [private]

Separate thread for power loading.

Definition at line 405 of file Dvbt1OfdmModComponent.cpp.

References _ampliFactor_, nFft_, powerFile_x, powerInterval_x, runPower_, and WAKEUPINTERVALMS.

Referenced by setup().

```
{
  int currentTick = 0, nextTick = currentTick;

  while(runPower_)
  {
    if(currentTick == nextTick)
    {
      // advance ticks
      nextTick = currentTick + 1000 * powerInterval_x / WAKEUPINTERVALMS;

      // open the file
      std::ifstream myfile(powerFile_x.c_str());
      if(myfile)
      {
        // read it line after line
        std::string line;
        int l = 0;
        while (std::getline(myfile, line) && l < nFft_)
        {
          // parse the line and get the dB, then convert it to linear
          float val = 0.0f;
          std::istringstream istr(line);
          istr.imbue(std::locale("C"));
          istr >> val;
          _ampliFactor_[l++] = powf(10.0F, val / 20.0F);
        }
      }
      else
      {
        LOG(LINFO) << "Power loading file '" << powerFile_x << "' not found";
      }
    }

    // this is to give responsivity
    boost::this_thread::sleep(boost::posix_time::milliseconds(WAKEUPINTERVALMS));
    currentTick++;
  }
}
```

7.10.4.10 void iris::phy::Dvbt1OfdmModComponent::process ( ) [virtual]

Main processing method.

Definition at line 141 of file Dvbt1OfdmModComponent.cpp.

References ampliFactor_, debug_x, fft_, fftBins_, inOffset_, inReg_, kMax_, multFactor_, nBlock_, nDelta_, nFft_, precorrFactor_, sampleRate_, and timeStamp_.

```
{
  // request input
  DataSet< Cplx > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = nBlock_ * ((insize + inOffset_) / kMax_);

  // request output
  DataSet< Cplx >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // fill the input register
  CplxVecIt outit = out->data.begin();
  for(CplxVecIt init = in->data.begin(); init < in->data.end(); init++)
  {
    // copy datum
    inReg_[inOffset_++] = *init;

    // trigger IFFT
    if(inOffset_ == kMax_)
    {
        int num_pos = kMax_ / 2 + 1;
        int num_neg = num_pos - 1;
        int neg_start = nFft_ - num_neg;

      // reset offset
      inOffset_ = 0;

      // copy positive frequencies
      for(int i = 0; i < num_pos; i++)
      {
        fftBins_[i] = inReg_[num_neg + i] * precorrFactor_[i] * ampliFactor_[i];
      }

      // copy negative frequencies
      for(int i = 0; i < num_neg; i++)
      {
        fftBins_[neg_start + i] = inReg_[i] * precorrFactor_[-num_neg + i] * ampliFactor_[-num_neg + i];
      }

      // set null frequencies
      for(int i = 0; i < nFft_ - kMax_; i++)
      {
        fftBins_[num_pos + i] = Cplx(0,0);
      }

      // call FFTW
      fftwf_execute(fft_);

      // apply multiplicative factor, for the power
      for(int i = 0; i < nFft_; i++)
      {
          fftBins_[i].real(fftBins_[i].real() * multFactor_);
          fftBins_[i].imag(fftBins_[i].imag() * multFactor_);
      }

      // copy to output
      CplxVecIt it = copy(&fftBins_[nFft_ - nDelta_], &fftBins_[nFft_], outit);
      copy(&fftBins_[0], &fftBins_[nFft_], it);

      outit += nBlock_;
    }
  }

  //set the timestamp and sample rate for the DataSets
  out->timeStamp = timeStamp_;
  out->sampleRate = sampleRate_;
  timeStamp_ += (double) outsize / sampleRate_;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.10.4.11 void iris::phy::Dvbt1OfdmModComponent::registerPorts ( ) [virtual]

Register the mapper ports with the IRIS system.

This component has one input that accept complex float values and one output that provides complex float values.

Definition at line 116 of file Dvbt1OfdmModComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< Cplx >::identifier);
  registerOutputPort("output1", TypeInfo< Cplx >::identifier);
}
```

7.10.4.12 void iris::phy::Dvbt1OfdmModComponent::setup ( ) [private]

Set up all needed constants.

Definition at line 295 of file Dvbt1OfdmModComponent.cpp.

References _ampliFactor_, _precorrFactor_, ampliFactor_, blackman_sinc(), dacSampleRate_x, deltaMode_x, fft_, fftBins_, fftReg_, frequency_response_modulus(), inOffset_, inReg_, kMax_, multFactor_, nBit_, nBlock_, nDelta_, nFft_, nMax_, ofdmMode_x, outPower_x, powerFile_x, powerProcedure_(), powerThread_, precorrFactor_, runPower_, sampleRate_, T1_RESAMPLE_ORDER, timeStamp_, and tpsNum_.

Referenced by initialize(), and parameterHasChanged().

```
{
  if(dacSampleRate_x == 0)
    dacSampleRate_x = 64.0e6/7.0;
  sampleRate_ = 64.0e6/7;
  timeStamp_ = 0;
  // clean registers
  switch(ofdmMode_x)
  {
    case 2048:
      tpsNum_ = 17;
      nMax_ = 1512;
      kMax_ = 1705;
      nBit_ = 11;
      nFft_ = 2048;
      break;
    case 4096:
      tpsNum_ = 34;
      nMax_ = 3024;
      kMax_ = 3409;
      nBit_ = 12;
      nFft_ = 4096;
      break;
    case 8192:
      tpsNum_ = 68;
      nMax_ = 6048;
      kMax_ = 6817;
      nBit_ = 13;
      nFft_ = 8192;
      break;
  }
  nDelta_ = (int) floor((double) nFft_ / (double) deltaMode_x);
  nBlock_ = nFft_ + nDelta_;
  inOffset_ = 0;
  inReg_.resize(kMax_);
  fftReg_.resize(nFft_);

    int num_pos = kMax_ / 2 + 1;
    int num_neg = num_pos - 1;
    float temp = 0.0F;

  // multiplicative factor
    float power = (1.0F * (float) nMax_ /* data carriers */
        + (16.0F / 9.0F) * (float) (kMax_ - nMax_ - tpsNum_) /* pilot carriers */
        + 1.0F * (float) tpsNum_ /* tps carriers */)
        / (float) nFft_;
  multFactor_ = (float) sqrt((outPower_x / 100.0F) / (power * (float)
  nFft_)) /
    3.0F;

  // linear precorrection
    double dtbase = (1 / (64.0e6/7.0)) / 100.0;
  int nbase = 0;
    double *hbase = blackman_sinc(&nbase, 1 / (64.0e6/7.0), dtbase,
  T1_RESAMPLE_ORDER);
  _precorrFactor_.resize(nFft_);
  precorrFactor_ = _precorrFactor_.begin() + nFft_ / 2;
    for(int i = -num_neg; i < num_pos; i++)
  {
    //printf("dacSampleRate_x = %f\n", dacSampleRate_x);
        if(dacSampleRate_x == 64.0e6/7.0)
        {
            precorrFactor_[i] = 1.0F; // no precorrection
        }
        else
            precorrFactor_[i] = (float) (1.0 /
  frequency_response_modulus(hbase,
              nbase, dtbase, (double) i * (64.0e6/7.0) / (double) nFft_));

        if (i == 0)
            temp = precorrFactor_[i];
    }

    // normalize
    for(int i = -num_neg; i < num_pos; i++)
        precorrFactor_[i] /= temp;
  free(hbase);

  // amplitude power loading factor
  _ampliFactor_.resize(nFft_);
  ampliFactor_ = _ampliFactor_.begin() + nFft_ / 2;
    for(int i = -num_neg; i < num_pos; i++)
    ampliFactor_[i] = 1.0F; // default powerloading
    if(!powerFile_x.empty())
  {
    // stop in case it's running
    if(powerThread_)
    {
      runPower_ = false;
      powerThread_->join();
      delete powerThread_;
    }

    // start thread
    runPower_ = true;
    powerThread_ = new boost::thread(boost::bind(&
  Dvbt1OfdmModComponent::powerProcedure_, this));
    }

  // Set up containers for FFTW
  fftBins_ = reinterpret_cast<Cplx*>(
      fftwf_malloc(sizeof(fftwf_complex) * ofdmMode_x));
  fill(&fftBins_[0], &fftBins_[ofdmMode_x], Cplx(0,0));
  fft_ = fftwf_plan_dft_1d(ofdmMode_x,
                           (fftwf_complex*)fftBins_,
                           (fftwf_complex*)fftBins_,
                           FFTW_BACKWARD,
                           FFTW_MEASURE);

}
```

7.10.4.13 double iris::phy::Dvbt1OfdmModComponent::sinc (double *x*) [private]

sin(x)/x function

Parameters

x	Input value

Returns

The sinc of the input

Definition at line 239 of file Dvbt1OfdmModComponent.cpp.

Referenced by blackman_sinc().

```
{
    return x == 0.0 ? 1.0 : (sin(x) / x);
}
```

7.10.5 Member Data Documentation

7.10.5.1 FloatVec iris::phy::Dvbt1OfdmModComponent::_ampliFactor_ [private]

Definition at line 210 of file Dvbt1OfdmModComponent.h.

Referenced by powerProcedure_(), and setup().

7.10.5.2 FloatVec iris::phy::Dvbt1OfdmModComponent::_precorrFactor_ [private]

Definition at line 208 of file Dvbt1OfdmModComponent.h.

Referenced by setup().

7.10.5.3 FloatVecIt iris::phy::Dvbt1OfdmModComponent::ampliFactor_ [private]

Definition at line 211 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.4 double iris::phy::Dvbt1OfdmModComponent::dacSampleRate_x [private]

Sampling rate used by the DAC.

Definition at line 187 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and setup().

7.10.5.5 bool iris::phy::Dvbt1OfdmModComponent::debug_x [private]

Debug flag (default = false)

Definition at line 183 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and process().

7.10.5.6 int iris::phy::Dvbt1OfdmModComponent::deltaMode_x [private]

Cyclic prefix ratio (default = 32)

Definition at line 185 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and setup().

7.10.5.7 fftwf_plan iris::phy::Dvbt1OfdmModComponent::fft_ [private]

Our FFT object pointer.

Definition at line 212 of file Dvbt1OfdmModComponent.h.

Referenced by destroy(), process(), and setup().

7.10.5.8 Cplx∗ iris::phy::Dvbt1OfdmModComponent::fftBins_ [private]

Allocated using fftwf_malloc (SIMD aligned)

Definition at line 213 of file Dvbt1OfdmModComponent.h.

Referenced by destroy(), process(), and setup().

7.10.5.9 CplxVec iris::phy::Dvbt1OfdmModComponent::fftReg_ [private]

Definition at line 202 of file Dvbt1OfdmModComponent.h.

Referenced by setup().

7.10.5.10 int iris::phy::Dvbt1OfdmModComponent::inOffset_ [private]

Definition at line 200 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.11 CplxVec iris::phy::Dvbt1OfdmModComponent::inReg_ [private]

Definition at line 201 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.12 int iris::phy::Dvbt1OfdmModComponent::kMax_ [private]

Definition at line 204 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.13 float iris::phy::Dvbt1OfdmModComponent::multFactor_ [private]

Definition at line 207 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.14 int iris::phy::Dvbt1OfdmModComponent::nBit_ [private]

Definition at line 206 of file Dvbt1OfdmModComponent.h.

Referenced by setup().

7.10.5.15 int iris::phy::Dvbt1OfdmModComponent::nBlock_ [private]

Definition at line 199 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.16 int iris::phy::Dvbt1OfdmModComponent::nDelta_ [private]

Definition at line 198 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.17 int iris::phy::Dvbt1OfdmModComponent::nFft_ [private]

Definition at line 197 of file Dvbt1OfdmModComponent.h.

Referenced by powerProcedure_(), process(), and setup().

7.10.5.18 int iris::phy::Dvbt1OfdmModComponent::nMax_ [private]

Definition at line 203 of file Dvbt1OfdmModComponent.h.

Referenced by setup().

7.10.5.19 int iris::phy::Dvbt1OfdmModComponent::ofdmMode_x [private]

OFDM mode (default = 2048)

Definition at line 184 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and setup().

7.10.5.20 float iris::phy::Dvbt1OfdmModComponent::outPower_x [private]

Output power indicator (default = 10)

Definition at line 186 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and setup().

7.10.5.21 std::string iris::phy::Dvbt1OfdmModComponent::powerFile_x [private]

Text file with power loading (default = none)

Definition at line 188 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), powerProcedure_(), and setup().

7.10.5.22 double iris::phy::Dvbt1OfdmModComponent::powerInterval_x [private]

Power update interval in seconds (default = 0)

Definition at line 189 of file Dvbt1OfdmModComponent.h.

Referenced by Dvbt1OfdmModComponent(), and powerProcedure_().

7.10.5.23 boost::thread* iris::phy::Dvbt1OfdmModComponent::powerThread_ [private]

Definition at line 216 of file Dvbt1OfdmModComponent.h.

Referenced by destroy(), and setup().

7.10.5.24 FloatVecIt iris::phy::Dvbt1OfdmModComponent::precorrFactor_ [private]

Definition at line 209 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.25 bool iris::phy::Dvbt1OfdmModComponent::runPower_ [private]

Definition at line 217 of file Dvbt1OfdmModComponent.h.

Referenced by destroy(), powerProcedure_(), and setup().

7.10.5.26 double iris::phy::Dvbt1OfdmModComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 195 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.27 double iris::phy::Dvbt1OfdmModComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 194 of file Dvbt1OfdmModComponent.h.

Referenced by process(), and setup().

7.10.5.28 int iris::phy::Dvbt1OfdmModComponent::tpsNum_ [private]

Definition at line 205 of file Dvbt1OfdmModComponent.h.

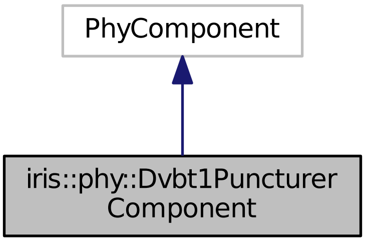
Referenced by setup().

7.11 iris::phy::Dvbt1PuncturerComponent Class Reference

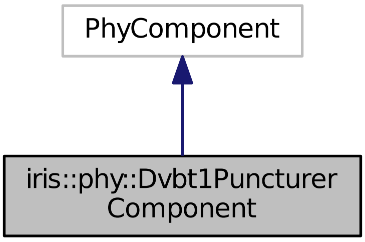
A DVB-T1 puncturer component.

```
#include <Dvbt1PuncturerComponent.h>
```

Inheritance diagram for iris::phy::Dvbt1PuncturerComponent:



Collaboration diagram for iris::phy::Dvbt1PuncturerComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.

- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1PuncturerComponent (std::string name)

  Default constructor.
- ∼Dvbt1PuncturerComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the puncturer ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up all puncturing basic sizes, reset offsets, clean registers.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int codeRate_x

  stream channel coding rate (default = 34)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int punOffset_

  Puncturing offset.
- uint8_t punRegister_ [14]

  Puncturing register (statically set to the maximum exected size)
- int punPeriodIn_
- int punPeriodOut_

  Input and output puncturing periods.

7.11.1 Detailed Description

A DVB-T1 puncturer component.

Dvbt1PuncturerComponent is the fifth block composing the DVB-T transmission chain. The purpose of the puncturer is that of achieving variable coding rate keeping fixed the properties and complexity of the main *mother* convolutional code, which sticks at a rate of $k/n = 1/2$. By properly removing convolutional encoded bits before transmission, one can still expect to take profit of the error correction capabilities of the convolutional decoder (Viterbi algorithm) at the receiving side, although having the possibility to change the overall coding rate to one of the values $r_c = 1/2, 2/3, 3/4, 5/6, 7/8$. The block operates by translating the puncturing matrices, that are given in the standard, into a periodic subsampling structure (a sort of nonequispaced bit decimation). Thus, a group of input bits (at the input periodicity) are read into a register, which is dumped into another shorter register, which in turn is read out (at the output periodicity).

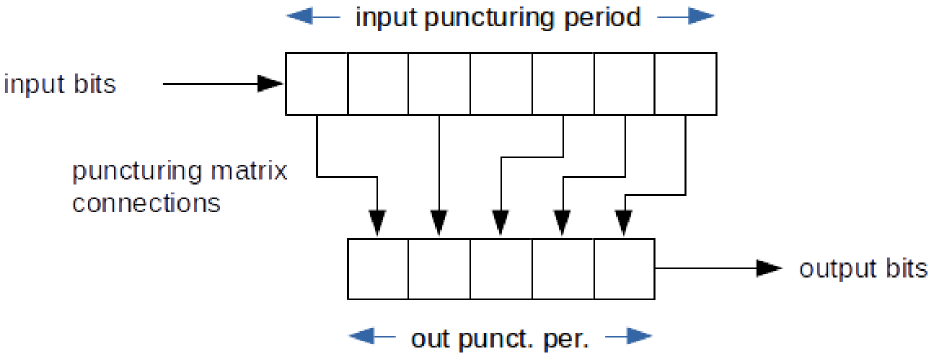


Figure 7.8: DVB-T puncturer.

This block accepts in input elements in uint8_t (bits) and generates in output bits (uint8_t).

There are two parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *coderate*: by default set to "34", this is used to select one of the five possible coding rates. The admitted values are "12", "23", "34", "56", and "78", which are easily recognizable as the real coding ratioes written without the separating slash.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at ETSI Publications Download Area
- S. Li, D. J. Costello, *Error Control Coding, Second Edition*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2004

Definition at line 84 of file Dvbt1PuncturerComponent.h.

7.11.2 Member Typedef Documentation

7.11.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1PuncturerComponent::ByteVec

A vector of bytes.

Definition at line 90 of file Dvbt1PuncturerComponent.h.

7.11.2.2 typedef ByteVec::iterator iris::phy::Dvbt1PuncturerComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 93 of file Dvbt1PuncturerComponent.h.

7.11.3 Constructor & Destructor Documentation

7.11.3.1 iris::phy::Dvbt1PuncturerComponent::Dvbt1PuncturerComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1PuncturerComponent.cpp.

References begin(), codeRate_x, debug_x, and end().

```
58   : PhyComponent(name,                          // component name
59                  "dvbt1puncturer",              // component type
60                  "A DVB-T1 puncturer component",   // description
61                  "Giuseppe Baruffa",            // author
62                  "0.1")                         // version
63     ,sampleRate_(0)
64     ,timeStamp_(0)
65     ,punOffset_(0)
66 {
67   registerParameter(
68     "debug", "Whether to output debug data",
69     "false", true, debug_x);
70 
71   int codearr[] = {12,23,34,56,78};
72   registerParameter(
73     "coderate", "Channel coding rate",
74     "34", true, codeRate_x, list<int>(begin(codearr),end(codearr)));
75 }
```

7.11.3.2 iris::phy::Dvbt1PuncturerComponent::∼Dvbt1PuncturerComponent ( )

Default destructor.

Just calls destroy().

Definition at line 80 of file Dvbt1PuncturerComponent.cpp.

References destroy().

```
81 {
82   destroy();
83 }
```

7.11.4 Member Function Documentation

7.11.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1PuncturerComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 123 of file Dvbt1PuncturerComponent.h.

Referenced by Dvbt1PuncturerComponent().

```
123 { return &arr[0]; }
```

7.11.4.2 void iris::phy::Dvbt1PuncturerComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 98 of file Dvbt1PuncturerComponent.cpp.

```
101 {
102   outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
103 }
```

7.11.4.3 void iris::phy::Dvbt1PuncturerComponent::destroy ( ) [private]

Destroy the component.

Definition at line 241 of file Dvbt1PuncturerComponent.cpp.

Referenced by parameterHasChanged(), and ~Dvbt1PuncturerComponent().

```
242 {
243 }
```

7.11.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1PuncturerComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 125 of file Dvbt1PuncturerComponent.h.

Referenced by Dvbt1PuncturerComponent().

```
125 { return &arr[0]+N; }
```

7.11.4.5 void iris::phy::Dvbt1PuncturerComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 108 of file Dvbt1PuncturerComponent.cpp.

References setup().

```
109 {
110   setup();
111 }
```

7.11.4.6 void iris::phy::Dvbt1PuncturerComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has one significant parameters

Definition at line 199 of file Dvbt1PuncturerComponent.cpp.

References destroy(), and setup().

```
200 {
201   if(name == "coderate")
202   {
203     destroy();
204     setup();
205   }
206 }
```

7.11.4.7 void iris::phy::Dvbt1PuncturerComponent::process ( ) [virtual]

Main processing method.

Definition at line 114 of file Dvbt1PuncturerComponent.cpp.

References codeRate_x, debug_x, punOffset_, punPeriodIn_, punPeriodOut_, and punRegister_.

```
{
  // request input
  DataSet< uint8_t >* in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = ((insize + punOffset_) / punPeriodIn_) * punPeriodOut_;

  // request output
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize + punOffset_ << "(" << insize << "+" << punOffset_ << ")/" << outsize;

  // iterate over input
  for(ByteVecIt init = in->data.begin(), outit = out->data.begin(); init < in->data.end(); init++)
  {
    // fill puncturing register
    punRegister_[punOffset_++] = *init;

    // trigger puncturing at the output
    if(punOffset_ == punPeriodIn_)
    {
      // reset offset
      punOffset_ = 0;

      // copy to output
      switch(codeRate_x)
      {
        // the puncturing matrices are hard-coded for all the five code rates
        case 12:
          *outit++ = punRegister_[0];
          *outit++ = punRegister_[1];
          break;
        case 23:
          *outit++ = punRegister_[0];
          *outit++ = punRegister_[1];
          *outit++ = punRegister_[3];
          break;
        case 34:
          *outit++ = punRegister_[0];
          *outit++ = punRegister_[1];
          *outit++ = punRegister_[3];
          *outit++ = punRegister_[4];
          break;
        case 56:
          *outit++ = punRegister_[0];
          *outit++ = punRegister_[1];
          *outit++ = punRegister_[3];
          *outit++ = punRegister_[4];
          *outit++ = punRegister_[7];
          *outit++ = punRegister_[8];
          break;
        case 78:
          *outit++ = punRegister_[0];
          *outit++ = punRegister_[1];
          *outit++ = punRegister_[3];
          *outit++ = punRegister_[5];
          *outit++ = punRegister_[7];
          *outit++ = punRegister_[8];
          *outit++ = punRegister_[11];
          *outit++ = punRegister_[12];
          break;
        default:
          LOG(LERROR) << "Invalid puncturing rate: " << codeRate_x;
      }
    }
  }

  // Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.11.4.8 void iris::phy::Dvbt1PuncturerComponent::registerPorts ( ) [virtual]

Register the puncturer ports with the IRIS system.

This component has one input that accepts bits (one bit per byte) and one output that provides punctured bits (one bit per byte).

Definition at line 89 of file Dvbt1PuncturerComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.11.4.9 void iris::phy::Dvbt1PuncturerComponent::setup ( ) [private]

Set up all puncturing basic sizes, reset offsets, clean registers.

Definition at line 209 of file Dvbt1PuncturerComponent.cpp.

References codeRate_x, punOffset_, punPeriodIn_, punPeriodOut_, and punRegister_.

Referenced by initialize(), and parameterHasChanged().

```
{
  switch(codeRate_x)
  {
    case 12:
      punPeriodIn_ = 2;
      punPeriodOut_ = 2;
      break;
    case 23:
      punPeriodIn_ = 4;
      punPeriodOut_ = 3;
      break;
    case 34:
      punPeriodIn_ = 6;
      punPeriodOut_ = 4;
      break;
    case 56:
      punPeriodIn_ = 10;
      punPeriodOut_ = 6;
      break;
    case 78:
      punPeriodIn_ = 14;
      punPeriodOut_ = 8;
      break;
    default:
      LOG(LERROR) << "Invalid puncturing rate: " << codeRate_x;
  }
  punOffset_ = 0;
  memset(punRegister_, 0, sizeof(punRegister_));
}
```

7.11.5 Member Data Documentation

7.11.5.1 int iris::phy::Dvbt1PuncturerComponent::codeRate_x [private]

stream channel coding rate (default = 34)

Definition at line 108 of file Dvbt1PuncturerComponent.h.

Referenced by Dvbt1PuncturerComponent(), process(), and setup().

7.11.5.2 bool iris::phy::Dvbt1PuncturerComponent::debug_x [private]

Debug flag (default = false)

Definition at line 107 of file Dvbt1PuncturerComponent.h.

Referenced by Dvbt1PuncturerComponent(), and process().

7.11.5.3 int iris::phy::Dvbt1PuncturerComponent::punOffset_ [private]

Puncturing offset.

Definition at line 116 of file Dvbt1PuncturerComponent.h.

Referenced by process(), and setup().

7.11.5.4 int iris::phy::Dvbt1PuncturerComponent::punPeriodIn_ [private]

Definition at line 119 of file Dvbt1PuncturerComponent.h.

Referenced by process(), and setup().

7.11.5.5 int iris::phy::Dvbt1PuncturerComponent::punPeriodOut_ [private]

Input and output puncturing periods.

Definition at line 119 of file Dvbt1PuncturerComponent.h.

Referenced by process(), and setup().

7.11.5.6 uint8_t iris::phy::Dvbt1PuncturerComponent::punRegister_[14] [private]

Puncturing register (statically set to the maximum exected size)

Definition at line 117 of file Dvbt1PuncturerComponent.h.

Referenced by process(), and setup().

7.11.5.7 double iris::phy::Dvbt1PuncturerComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 114 of file Dvbt1PuncturerComponent.h.

7.11.5.8 double iris::phy::Dvbt1PuncturerComponent::timeStamp_ [private]

Timestamp of current frame.

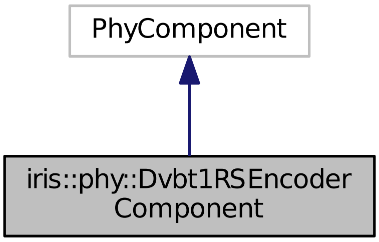
Definition at line 113 of file Dvbt1PuncturerComponent.h.

7.12 iris::phy::Dvbt1RSEncoderComponent Class Reference

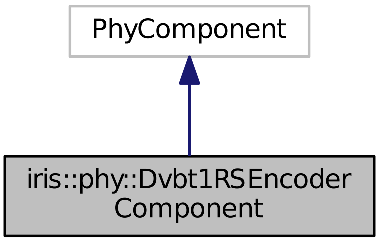
A DVB-T1 R-S Encoder component.

```
#include <Dvbt1RSEncoderComponent.h>
```

Inheritance diagram for iris::phy::Dvbt1RSEncoderComponent:



Collaboration diagram for iris::phy::Dvbt1RSEncoderComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1RSEncoderComponent (std::string name)

  Default constructor.
- ∼Dvbt1RSEncoderComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the encoder ports with the IRIS system.

- virtual void initialize ()

  Initialize the component.

- virtual void process ()

  Main processing method.

- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up offsets and clean variables.

- void destroy ()

  Destroy the component.

- int packetEncode (unsigned char ∗data, unsigned char ∗bb)

  Encodes a single data packet.

- int modnn (int x)

  Computes the modulo-255 of a number.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.

- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)

- double timeStamp_

  Timestamp of current frame.

- double sampleRate_

  Sample rate of current frame.

- uint8_t rsCodeWord_ [T1_NN]

  Nonshortened codeword.

- int tsOffset_

  Current offset in TS input.

Static Private Attributes

- static int index_ [256]

  LUT containing the base α logarithm of the field elements.

- static int alpha_ [256]

  LUT containing the powers of α.

- static int gg_ [17]

  R-S code generator polynomial.

7.12.1 Detailed Description

A DVB-T1 R-S Encoder component.

Dvbt1RSEncoderComponent is the second block composing the DVB-T transmission chain. This block is a non-binary Reed-Solomon (R-S) encoder operating on the Galois field GF(2^8) of 256 elements. Every element in the field is either 0 or an integer power of a primitive element α; the field is generated by the primitive polynomial $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. The code generator polynomial, instead, is generated to have as roots all the first 16 powers (0 to 15) of the primitive element $\alpha = 2$,as

$$g(x) = (x+\alpha^0)(x+\alpha^1)\ldots\left(x+\alpha^{15}\right) .$$

The encoder computes the remainder of the division of the message polynomial $m(x)$, of 239 bytes, by the generator polynomial $g(x)$, and considers this as the parity polynomial $p(x)$, of 16 bytes. The codeword is then composed py appending the parity polynomial and the message polynomial together (255 bytes), as

$$c(x) = p(x) + x^{16} m(x) .$$

This code is capable of correcting $t = 8$ errated bytes in every codeword. Actually, DVB-T uses shortened codewords of 204 bytes, generated by messages of 188 bytes prepended by a string of 51 zero bytes. The encoder itself can be implemented with a feedback shift register, operating in GF(2^8). Please note that the codewords are message-first parity-last ordered.

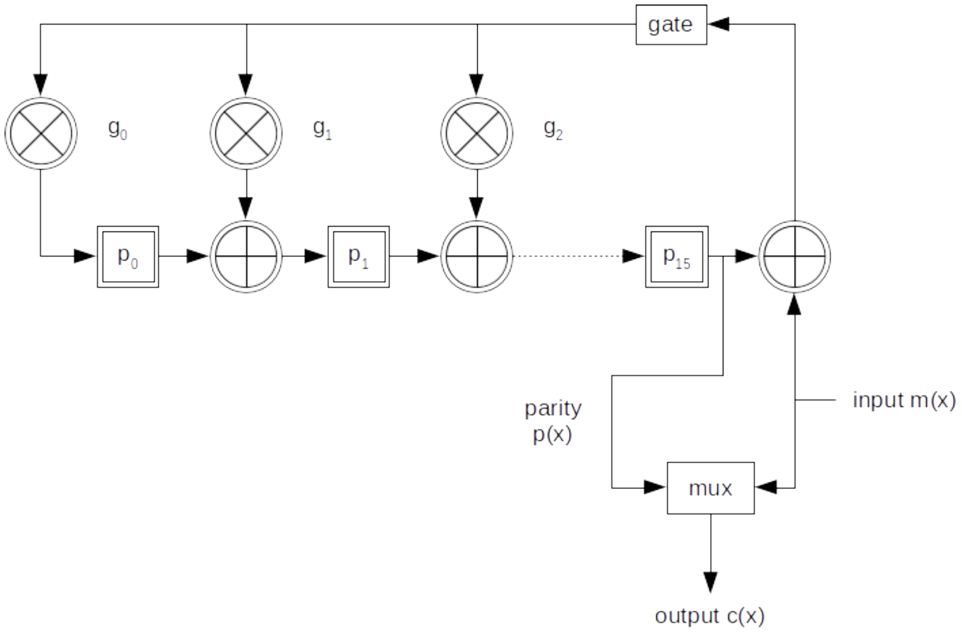


Figure 7.9: DVB-T shortened R-S encoder.

Differently from the simpler multiply-and-add operations in the binary Galois field GF(2), in this case we must recur to byte operators, which are practically implemented with look-up tables that perform exponentiation and logarithm of the GF(2^8) elements. Particular care is taken to consider the zero element.

There is only one parameter that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at `ETSI Publications Download Area`

- S. Li, D. J. Costello, *Error Control Coding, Second Edition*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2004

Definition at line 121 of file Dvbt1RSEncoderComponent.h.

7.12.2 Member Typedef Documentation

7.12.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1RSEncoderComponent::ByteVec

A vector of bytes.

Definition at line 127 of file Dvbt1RSEncoderComponent.h.

7.12.2.2 typedef ByteVec::iterator iris::phy::Dvbt1RSEncoderComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 130 of file Dvbt1RSEncoderComponent.h.

7.12.3 Constructor & Destructor Documentation

7.12.3.1 iris::phy::Dvbt1RSEncoderComponent::Dvbt1RSEncoderComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1RSEncoderComponent.cpp.

References debug_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1rsencoder",                 // component type
                "A DVB-T1 R-S encoder component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,tsOffset_(0)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);
}
```

7.12.3.2 iris::phy::Dvbt1RSEncoderComponent::∼Dvbt1RSEncoderComponent ( )

Default destructor.

Just calls destroy().

Definition at line 75 of file Dvbt1RSEncoderComponent.cpp.

References destroy().

```
{
  destroy();
}
```

7.12.4 Member Function Documentation

7.12.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1RSEncoderComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 172 of file Dvbt1RSEncoderComponent.h.

```
{ return &arr[0]; }
```

7.12.4.2 void iris::phy::Dvbt1RSEncoderComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 93 of file Dvbt1RSEncoderComponent.cpp.

```
{
  outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
}
```

7.12.4.3 void iris::phy::Dvbt1RSEncoderComponent::destroy ( ) [private]

Destroy the component.

Definition at line 264 of file Dvbt1RSEncoderComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1RSEncoderComponent().

```
{
}
```

7.12.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1RSEncoderComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 174 of file Dvbt1RSEncoderComponent.h.

```
{ return &arr[0]+N; }
```

7.12.4.5 void iris::phy::Dvbt1RSEncoderComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 103 of file Dvbt1RSEncoderComponent.cpp.

References setup().

```
{
  setup();
}
```

7.12.4.6 int iris::phy::Dvbt1RSEncoderComponent::modnn (int *x*) [inline], [private]

Computes the modulo-255 of a number.

Definition at line 151 of file Dvbt1RSEncoderComponent.h.

References T1_MM, and T1_NN.

Referenced by packetEncode().

```
  {
      while (x >= T1_NN) {
          x -= T1_NN;
          x = (x >> T1_MM) + (x & T1_NN);
      }
      return x;
  };
```

7.12.4.7 int iris::phy::Dvbt1RSEncoderComponent::packetEncode (unsigned char ∗ *data,* unsigned char ∗ *bb*) [private]

Encodes a single data packet.

Provides, at the output, a systematic encoded codeword where the first 188 bytes are the message, and the last 16 bytes are the parity.

Definition at line 160 of file Dvbt1RSEncoderComponent.cpp.

References alpha_, gg_, index_, modnn(), T1_A0, T1_CLEAR, T1_KK, T1_NN, and T1_NN_KK.

Referenced by process().

```
{
    T1_CLEAR(bb,T1_NN-T1_KK);
    for(int i = T1_KK - 1; i >= 0; i--)
    {
        int feedback = index_[data[i] ^ bb[T1_NN_KK - 1]]; // feedback term
        if(feedback != T1_A0)
        {
          // feedback term is non-zero
            for(int j = T1_NN_KK - 1; j > 0; j--)
                if(gg_[j] != T1_A0)
                    bb[j] = bb[j - 1] ^ alpha_[modnn(gg_[j] + feedback)];
                else
                    bb[j] = bb[j - 1];
            bb[0] = alpha_[modnn(gg_[0] + feedback)]; // terminal connection
        }
        else
        {
          // feedback term is zero
            for(int j = T1_NN_KK - 1; j > 0; j--)
                bb[j] = bb[j - 1];
            bb[0] = 0;
        }
    }
    return 0;
}
```

7.12.4.8 void iris::phy::Dvbt1RSEncoderComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has no significant parameters

Definition at line 246 of file Dvbt1RSEncoderComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "???")
  {
    destroy();
    setup();
  }
}
```

7.12.4.9 void iris::phy::Dvbt1RSEncoderComponent::process ( ) [virtual]

Main processing method.

Definition at line 188 of file Dvbt1RSEncoderComponent.cpp.

References debug_x, packetEncode(), RS_PACKET_SIZE, rsCodeWord_, T1_KK, T1_NN, T1_NN_KK, TS_PACKET_SIZE, and tsOffset_.

```
{
  // request input
  DataSet< uint8_t >* in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int numpacks = (insize + tsOffset_) / TS_PACKET_SIZE;
  int outsize = numpacks * RS_PACKET_SIZE;

  // request output
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize + tsOffset_ << "(" << insize << "+" <<
  tsOffset_ << ")/" << outsize;

  // fill the messagewords
  for(ByteVecIt init = in->data.begin(), outit = out->data.begin(); init < in->data.end(); init++)
  {

    // copy in reverse order
    rsCodeWord_[TS_PACKET_SIZE - 1 - tsOffset_] = *init;

    // trigger encoding
    if(++tsOffset_ == TS_PACKET_SIZE)
    {
      int status = packetEncode(rsCodeWord_, rsCodeWord_ +
  T1_KK);
            if (status)
                LOG(LERROR) << "Problem encoding a R-S word";

      // copy information part
      for(int b = 0; b < TS_PACKET_SIZE; b++, outit++)
        *outit = rsCodeWord_[TS_PACKET_SIZE - 1 - b];

            // copy parity part
            for(int b = 0; b < T1_NN_KK; b++, outit++)
                *outit = rsCodeWord_[T1_NN - 1 - b];

      // reset TS pointer
      tsOffset_ = 0;
    }
  }

  //Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.12.4.10 void iris::phy::Dvbt1RSEncoderComponent::registerPorts ( ) [virtual]

Register the encoder ports with the IRIS system.

This component has one input that accepts bytes and one output that provides encoded bytes.

Definition at line 84 of file Dvbt1RSEncoderComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.12.4.11 void iris::phy::Dvbt1RSEncoderComponent::setup ( ) [private]

Set up offsets and clean variables.

Definition at line 256 of file Dvbt1RSEncoderComponent.cpp.

References rsCodeWord_, T1_NN, and tsOffset_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // clean
  memset(rsCodeWord_, 0, T1_NN);
  tsOffset_ = 0;
}
```

7.12.5 Member Data Documentation

7.12.5.1 int iris::phy::Dvbt1RSEncoderComponent::alpha_ [static], [private]

Initial value:

```
=
{
    1, 2, 4, 8, 16, 32, 64, 128, 29, 58, 116, 232, 205, 135, 19, 38,
    76, 152, 45, 90, 180, 117, 234, 201, 143, 3, 6, 12, 24, 48, 96, 192,
    157, 39, 78, 156, 37, 74, 148, 53, 106, 212, 181, 119, 238, 193, 159, 35,
    70, 140, 5, 10, 20, 40, 80, 160, 93, 186, 105, 210, 185, 111, 222, 161,
    95, 190, 97, 194, 153, 47, 94, 188, 101, 202, 137, 15, 30, 60, 120, 240,
    253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223, 163, 91, 182, 113, 226,
    217, 175, 67, 134, 17, 34, 68, 136, 13, 26, 52, 104, 208, 189, 103, 206,
    129, 31, 62, 124, 248, 237, 199, 147, 59, 118, 236, 197, 151, 51, 102, 204,
    133, 23, 46, 92, 184, 109, 218, 169, 79, 158, 33, 66, 132, 21, 42, 84,
    168, 77, 154, 41, 82, 164, 85, 170, 73, 146, 57, 114, 228, 213, 183, 115,
    230, 209, 191, 99, 198, 145, 63, 126, 252, 229, 215, 179, 123, 246, 241, 255,
    227, 219, 171, 75, 150, 49, 98, 196, 149, 55, 110, 220, 165, 87, 174, 65,
    130, 25, 50, 100, 200, 141, 7, 14, 28, 56, 112, 224, 221, 167, 83, 166,
    81, 162, 89, 178, 121, 242, 249, 239, 195, 155, 43, 86, 172, 69, 138, 9,
    18, 36, 72, 144, 61, 122, 244, 245, 247, 243, 251, 235, 203, 139, 11, 22,
    44, 88, 176, 125, 250, 233, 207, 131, 27, 54, 108, 216, 173, 71, 142, 0
}
```

LUT containing the powers of α.

Definition at line 167 of file Dvbt1RSEncoderComponent.h.

Referenced by packetEncode().

7.12.5.2 bool iris::phy::Dvbt1RSEncoderComponent::debug_x [private]

Debug flag (default = false)

Definition at line 144 of file Dvbt1RSEncoderComponent.h.

Referenced by Dvbt1RSEncoderComponent(), and process().

7.12.5.3 int iris::phy::Dvbt1RSEncoderComponent::gg_ [static], [private]

Initial value:

```
=
{
  120, 225, 194, 182, 169, 147, 191, 91, 3, 76, 161, 102, 109, 107, 104, 120, 0
}
```

R-S code generator polynomial.

Definition at line 168 of file Dvbt1RSEncoderComponent.h.

Referenced by packetEncode().

7.12.5.4 int iris::phy::Dvbt1RSEncoderComponent::index_ [static], [private]

Initial value:

```
=
{
    255, 0, 1, 25, 2, 50, 26, 198, 3, 223, 51, 238, 27, 104, 199, 75,
    4, 100, 224, 14, 52, 141, 239, 129, 28, 193, 105, 248, 200, 8, 76, 113,
    5, 138, 101, 47, 225, 36, 15, 33, 53, 147, 142, 218, 240, 18, 130, 69,
    29, 181, 194, 125, 106, 39, 249, 185, 201, 154, 9, 120, 77, 228, 114, 166,
    6, 191, 139, 98, 102, 221, 48, 253, 226, 152, 37, 179, 16, 145, 34, 136,
    54, 208, 148, 206, 143, 150, 219, 189, 241, 210, 19, 92, 131, 56, 70, 64,
    30, 66, 182, 163, 195, 72, 126, 110, 107, 58, 40, 84, 250, 133, 186, 61,
    202, 94, 155, 159, 10, 21, 121, 43, 78, 212, 229, 172, 115, 243, 167, 87,
    7, 112, 192, 247, 140, 128, 99, 13, 103, 74, 222, 237, 49, 197, 254, 24,
    227, 165, 153, 119, 38, 184, 180, 124, 17, 68, 146, 217, 35, 32, 137, 46,
    55, 63, 209, 91, 149, 188, 207, 205, 144, 135, 151, 178, 220, 252, 190, 97,
    242, 86, 211, 171, 20, 42, 93, 158, 132, 60, 57, 83, 71, 109, 65, 162,
    31, 45, 67, 216, 183, 123, 164, 118, 196, 23, 73, 236, 127, 12, 111, 246,
    108, 161, 59, 82, 41, 157, 85, 170, 251, 96, 134, 177, 187, 204, 62, 90,
    203, 89, 95, 176, 156, 169, 160, 81, 11, 245, 22, 235, 122, 117, 44, 215,
    79, 174, 213, 233, 230, 231, 173, 232, 116, 214, 244, 234, 168, 80, 88, 175
}
```

LUT containing the base α logarithm of the field elements.

Definition at line 165 of file Dvbt1RSEncoderComponent.h.

Referenced by packetEncode().

7.12.5.5 uint8_t iris::phy::Dvbt1RSEncoderComponent::rsCodeWord_[T1_NN] [private]

Nonshortened codeword.

Definition at line 163 of file Dvbt1RSEncoderComponent.h.

Referenced by process(), and setup().

7.12.5.6 double iris::phy::Dvbt1RSEncoderComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 161 of file Dvbt1RSEncoderComponent.h.

7.12.5.7 double iris::phy::Dvbt1RSEncoderComponent::timeStamp_ [private]

Timestamp of current frame.

Definition at line 158 of file Dvbt1RSEncoderComponent.h.

7.12.5.8 int iris::phy::Dvbt1RSEncoderComponent::tsOffset_ [private]

Current offset in TS input.

Definition at line 164 of file Dvbt1RSEncoderComponent.h.

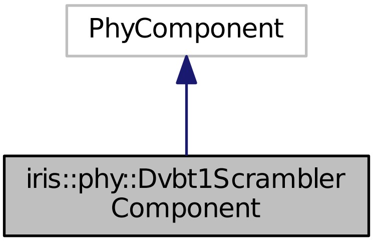
Referenced by process(), and setup().

7.13 iris::phy::Dvbt1ScramblerComponent Class Reference

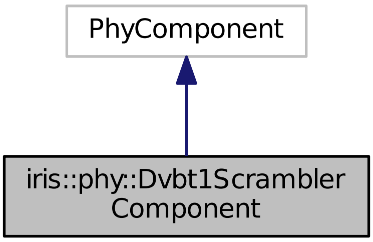
A DVB-T energy dispersal component.

`#include <Dvbt1ScramblerComponent.h>`

Inheritance diagram for iris::phy::Dvbt1ScramblerComponent:



Collaboration diagram for iris::phy::Dvbt1ScramblerComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1ScramblerComponent (std::string name)

  Default constructor.

- ∼Dvbt1ScramblerComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the scrambler ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up counters, offsets, etc.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- double reportInterval_x

  Reporting interval in seconds (default = 0)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.
- int scramblerOffset_

  Current scrambling offset.
- boost::posix_time::ptime start_

  Timestamp used for frame error rate reports.
- uint64_t doneBytes_

  currently processed bytes

Static Private Attributes

- static uint8_t scramblerPrbs_ [1504]

  Scrambling PRBS bytes.

7.13.1 Detailed Description

A DVB-T energy dispersal component.

Dvbt1ScramblerComponent is the first block composing the DVB-T transmission chain. This block takes an MPEG-2 Transport Stream (TS) of data bytes in uint8_t format and outputs a scrambled stream of uint8_t data. Per the DVB-T standard, the PRBS generator polynomial is $1+X^{14}+X^{15}$. The scrambler operates on a group of eight TS packets: each packet is 188-byte long and begins with the SYNC byte, 0x47. The PRBS register is loaded with the sequence "100101010000000" and is shift-enabled after the eighth bit, thus the ninth bit is the first to be scrambled. The other 7 SYNC bytes in the group are then bitwise-inverted to 0xB8, so as to provide a viable means for recovering scrambling synchrony at the receiver. The process is then repeated for the following groups of eight packets. The full period of the scrambling sequence is thus of 188 $*$ 8 - 1 = 1503 bytes.

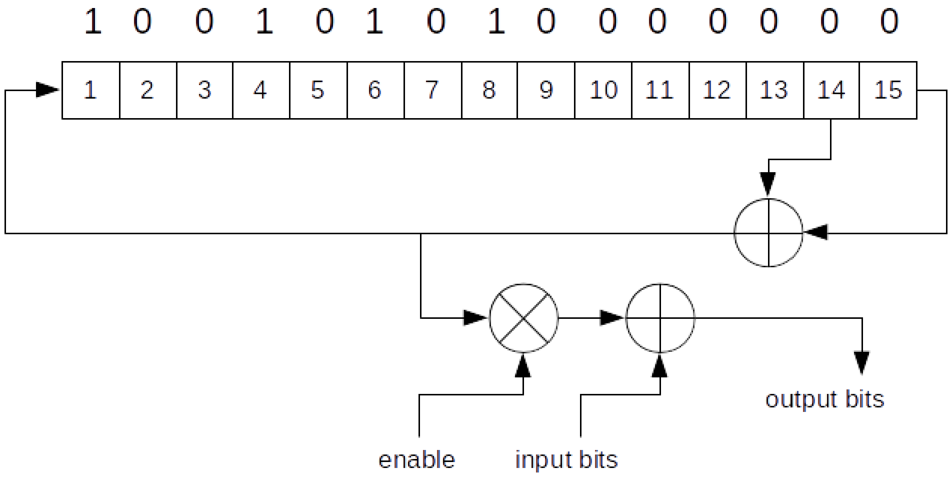


Figure 7.10: DVB-T energy dispersal.

There are two parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *reportinterval*: by default set to "0", which means it is disabled. If a number greater than zero is used, then it will be the number of seconds between which the block reports the computed processing speed. This can be useful to benchmark on-the-fly the processing speed of a complete DVB-T modulator graph that uses this block as source: if the graph is free-running, i.e., not terminated into an USRP block or similar, it will provide the maximum TS bitrate that the CPU is capable to process. Differently, if terminated into an USRP, this can be used to verify if the expected bitrate value (for that particular combination of DVB-T modulation and coding parameters) is honored.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at ETSI Publications Download Area

Definition at line 87 of file Dvbt1ScramblerComponent.h.

7.13.2 Member Typedef Documentation

7.13.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1ScramblerComponent::ByteVec

A vector of bytes.

Definition at line 93 of file Dvbt1ScramblerComponent.h.

7.13.2.2 typedef ByteVec::iterator iris::phy::Dvbt1ScramblerComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 96 of file Dvbt1ScramblerComponent.h.

7.13.3 Constructor & Destructor Documentation

7.13.3.1 iris::phy::Dvbt1ScramblerComponent::Dvbt1ScramblerComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 58 of file Dvbt1ScramblerComponent.cpp.

References debug_x, and reportInterval_x.

```
59   : PhyComponent(name,                          // component name
60                  "dvbt1scrambler",              // component type
61                  "A DVB-T1 scrambler component", // description
62                  "Giuseppe Baruffa",            // author
63                  "0.1")                         // version
64     ,sampleRate_(0)
65     ,timeStamp_(0)
66     ,scramblerOffset_(0)
67 {
68   registerParameter(
69     "debug", "Whether to output debug data",
70     "false", true, debug_x);
71
72   registerParameter(
73     "reportinterval", "Report interval in seconds",
74     "0", true, reportInterval_x);
75 }
```

7.13.3.2 iris::phy::Dvbt1ScramblerComponent::∼Dvbt1ScramblerComponent ( )

Default destructor.

Just calls destroy().

Definition at line 80 of file Dvbt1ScramblerComponent.cpp.

References destroy().

```
81 {
82   destroy();
83 }
```

7.13.4 Member Function Documentation

7.13.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ScramblerComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 125 of file Dvbt1ScramblerComponent.h.

```
125 { return &arr[0]; }
```

7.13.4.2 void iris::phy::Dvbt1ScramblerComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 98 of file Dvbt1ScramblerComponent.cpp.

```
101 {
102   outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
103 }
```

7.13.4.3 void iris::phy::Dvbt1ScramblerComponent::destroy ( ) [private]

Destroy the component.

Definition at line 280 of file Dvbt1ScramblerComponent.cpp.

Referenced by parameterHasChanged(), and ∼Dvbt1ScramblerComponent().

```
281 {
282 }
```

7.13.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1ScramblerComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 127 of file Dvbt1ScramblerComponent.h.

```
127 { return &arr[0]+N; }
```

7.13.4.5 void iris::phy::Dvbt1ScramblerComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 108 of file Dvbt1ScramblerComponent.cpp.

References setup().

```
109 {
110   setup();
111 }
```

7.13.4.6 void iris::phy::Dvbt1ScramblerComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block is not to be reset if parameters change

Definition at line 262 of file Dvbt1ScramblerComponent.cpp.

References destroy(), and setup().

```
263 {
264   if(name == "???")
265   {
266     destroy();
267     setup();
268   }
269 }
```

7.13.4.7 void iris::phy::Dvbt1ScramblerComponent::process ( ) [virtual]

Main processing method.

Definition at line 212 of file Dvbt1ScramblerComponent.cpp.

References debug_x, doneBytes_, reportInterval_x, scramblerOffset_, scramblerPrbs_, and start_.

```
{
  // request input
  DataSet< uint8_t >* in = NULL;
  getInputDataSet("input1", in);
  int size = in ? (int) in->data.size() : 0;

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << size << "/" << size;

  // request output - same size
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, size);

  // do the scrambling using the static array above
  for(ByteVecIt init = in->data.begin(), outit = out->data.begin(); init < in->data.end(); init++, outit++) {
    *outit = *init ^ scramblerPrbs_[scramblerOffset_];
    if (++scramblerOffset_ == 1504)
      scramblerOffset_ = 0;
  }

  // Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);

  // print the calculated bitrate
  if(reportInterval_x)
  {
    ptime t = microsec_clock::local_time(); // current time
    doneBytes_ += size; // increase processed bytes since last report
    time_duration delta = t-start_; // time elapsed from last report
    if(delta > seconds(reportInterval_x))
    {
      // interval is triggered, compute speed and report
      LOG(LINFO) << "Current TS bitrate: " << 8.0 * (double) doneBytes_ / (delta.total_microseconds()) << " Mbps";
      // reset counters
      start_ = t;
      doneBytes_ = 0;
    }
  }
}
```

7.13.4.8 void iris::phy::Dvbt1ScramblerComponent::registerPorts ( ) [virtual]

Register the scrambler ports with the IRIS system.

This component has one input that accepts TS bytes and one output that provides scrambled bytes.

Definition at line 89 of file Dvbt1ScramblerComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.13.4.9 void iris::phy::Dvbt1ScramblerComponent::setup ( ) [private]

Set up counters, offsets, etc.

Definition at line 272 of file Dvbt1ScramblerComponent.cpp.

References doneBytes_, scramblerOffset_, and start_.

Referenced by initialize(), and parameterHasChanged().

```
{
  scramblerOffset_ = 0;
  start_ = microsec_clock::local_time();
  doneBytes_ = 0L;
}
```

7.13.5 Member Data Documentation

7.13.5.1 bool iris::phy::Dvbt1ScramblerComponent::debug_x [private]

Debug flag (default = false)

Definition at line 110 of file Dvbt1ScramblerComponent.h.

Referenced by Dvbt1ScramblerComponent(), and process().

7.13.5.2 uint64_t iris::phy::Dvbt1ScramblerComponent::doneBytes_ [private]

currently processed bytes

Definition at line 121 of file Dvbt1ScramblerComponent.h.

Referenced by process(), and setup().

7.13.5.3 double iris::phy::Dvbt1ScramblerComponent::reportInterval_x [private]

Reporting interval in seconds (default = 0)

Definition at line 111 of file Dvbt1ScramblerComponent.h.

Referenced by Dvbt1ScramblerComponent(), and process().

7.13.5.4 double iris::phy::Dvbt1ScramblerComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 117 of file Dvbt1ScramblerComponent.h.

7.13.5.5 int iris::phy::Dvbt1ScramblerComponent::scramblerOffset_ [private]

Current scrambling offset.

Definition at line 119 of file Dvbt1ScramblerComponent.h.

Referenced by process(), and setup().

7.13.5.6 uint8_t iris::phy::Dvbt1ScramblerComponent::scramblerPrbs_ [static], [private]

Scrambling PRBS bytes.

Scrambling sequence bytes.

Definition at line 129 of file Dvbt1ScramblerComponent.h.

Referenced by process().

7.13.5.7 boost::posix_time::ptime iris::phy::Dvbt1ScramblerComponent::start_ [private]

Timestamp used for frame error rate reports.

Definition at line 120 of file Dvbt1ScramblerComponent.h.

Referenced by process(), and setup().

7.13.5.8 double iris::phy::Dvbt1ScramblerComponent::timeStamp_ [private]

Timestamp of current frame.

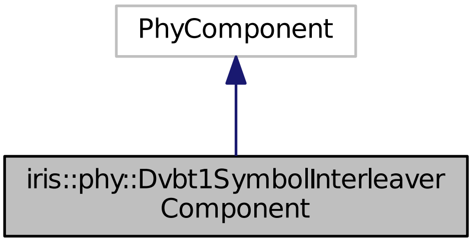
Definition at line 116 of file Dvbt1ScramblerComponent.h.

7.14 iris::phy::Dvbt1SymbolInterleaverComponent Class Reference

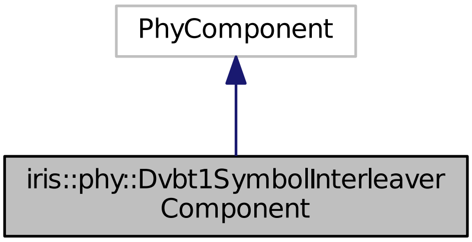
A DVB-T1 symbol interleaver component.

`#include <Dvbt1SymbolInterleaverComponent.h>`

Inheritance diagram for iris::phy::Dvbt1SymbolInterleaverComponent:



Collaboration diagram for iris::phy::Dvbt1SymbolInterleaverComponent:



Public Types

- typedef std::vector< uint8_t > ByteVec

  A vector of bytes.
- typedef ByteVec::iterator ByteVecIt

  An iterator for a vector of bytes.

Public Member Functions

- Dvbt1SymbolInterleaverComponent (std::string name)

  Default constructor.
- ∼Dvbt1SymbolInterleaverComponent ()

  Default destructor.
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)

  Calculate the output port types for the IRIS system.
- virtual void registerPorts ()

  Register the interleaver ports with the IRIS system.
- virtual void initialize ()

  Initialize the component.
- virtual void process ()

  Main processing method.
- virtual void parameterHasChanged (std::string name)

  Actions taken when the parameters change.

Private Member Functions

- void setup ()

  Set up the register space and initialize.
- void destroy ()

  Destroy the component.

Static Private Member Functions

- template<typename T , size_t N>
  static T ∗ begin (T(&arr)[N])

  Useful templates.
- template<typename T , size_t N>
  static T ∗ end (T(&arr)[N])

Private Attributes

- bool debug_x

  Debug flag (default = false)
- int ofdmMode_x

  OFDM mode (default = 2048)
- double timeStamp_

  Timestamp of current frame.
- double sampleRate_

  Sample rate of current frame.

- int siOffset_

  Interleaving offset.
- int siLength_

  Interleaving register length.
- uint8_t * siRegister_

  Actual interleaving register.
- int eo_

  Even/odd numbered OFDM block.

Static Private Attributes

- static int H_2K_ [1512]

  Interleaving addresses for 2K.
- static int H_4K_ [3024]

  Interleaving addresses for 4K.
- static int H_8K_ [6048]

  Interleaving addresses for 8K.

7.14.1 Detailed Description

A DVB-T1 symbol interleaver component.

Dvbt1SymbolInterleaverComponent is the seventh block composing the DVB-T transmission chain. Its purpose, together with the bit interleaver, is that of reordering the channel encoded bits in order to convert the possible error bursts arising from the communication on the physical channel (due to impulsive noise, multipath, fading) into well-separated single-error events. This way, the channel decoders at the RX side (Viterbi and Reed-Solomon decoder) are able to perform at their best theoretical limit in white Gaussian noise (WGN) conditions.

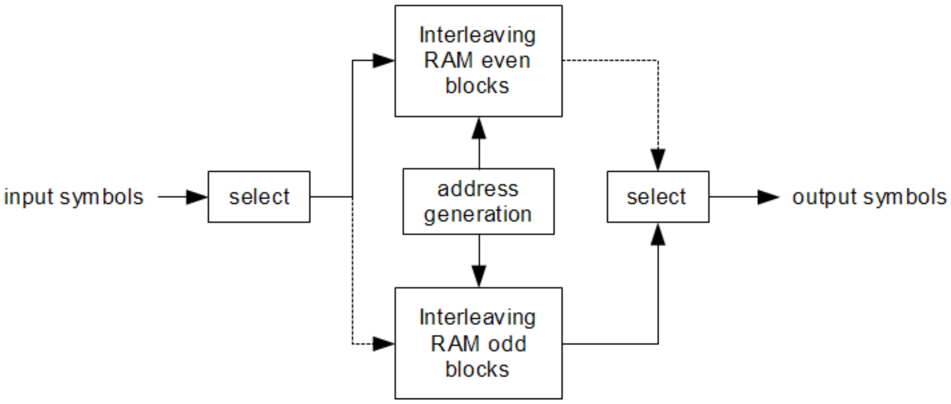


Figure 7.11: DVB-T symbol interleaver.

The symbol interleaver is a block-based interleaver, i.e., a block of consecutive symbols is written in the interleaving RAM, and then the same symbols are read into an output block with pseudo-random read addresses. Every output block of symbols is mapped into an OFDM block. For instance, in the 8K case, the symbol interleaver memory is of 6048 cells (1512 for the 2K case). The pseudo- random interleaving law is generated by means of linear feedback registers, whose state is turned into a valid interleaving address with the help of a bit mapping between the register bits and the addressing bits. Every second interleaving block, the interleaving law is exchanged between the reading and writing processes. In the practical implementation used in IRIS, however, the interleaving addresses are statically embedded in the source files, and a simple address mapping law is applied.

This block accepts in input elements in uint8_t (v-bit symbols) and generates in output v-bit symbols (uint8_t).

There are three parameters that can be changed in the XML configuration file:

- *debug*: by default set to "false", is used to print some small debugging information for the interested developer.
- *ofdmmode*: by default set to "2048", this is used to select one of the three possible OFDM modes. The admitted values are "2048", "4096", "8192", respectively for 2K, 4K (DVB-H, unused), and 8K.

References

- ETSI Standard: *EN 300 744 V1.5.1, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, available at ETSI Publications Download Area

Definition at line 89 of file Dvbt1SymbolInterleaverComponent.h.

7.14.2 Member Typedef Documentation

7.14.2.1 typedef std::vector<uint8_t> iris::phy::Dvbt1SymbolInterleaverComponent::ByteVec

A vector of bytes.

Definition at line 95 of file Dvbt1SymbolInterleaverComponent.h.

7.14.2.2 typedef ByteVec::iterator iris::phy::Dvbt1SymbolInterleaverComponent::ByteVecIt

An iterator for a vector of bytes.

Definition at line 98 of file Dvbt1SymbolInterleaverComponent.h.

7.14.3 Constructor & Destructor Documentation

7.14.3.1 iris::phy::Dvbt1SymbolInterleaverComponent::Dvbt1SymbolInterleaverComponent (std::string *name*)

Default constructor.

Registers the block parameters and initializes some variables

Definition at line 57 of file Dvbt1SymbolInterleaverComponent.cpp.

References begin(), debug_x, end(), and ofdmMode_x.

```
  : PhyComponent(name,                            // component name
                "dvbt1symbolinterleaver",                 // component type
                "A DVB-T1 symbol interleaver component", // description
                "Giuseppe Baruffa",               // author
                "0.1")                            // version
    ,sampleRate_(0)
    ,timeStamp_(0)
    ,siRegister_(NULL)
{
  registerParameter(
    "debug", "Whether to output debug data",
    "false", true, debug_x);

  int ofdmarr[] = {2048,4096,8192};
  registerParameter(
    "ofdmmode", "OFDM mode",
    "2048", true, ofdmMode_x, list<int>(begin(ofdmarr),end(ofdmarr)));
}
```

7.14.3.2 iris::phy::Dvbt1SymbolInterleaverComponent::∼Dvbt1SymbolInterleaverComponent ( )

Default destructor.

Just calls destroy().

Definition at line 80 of file Dvbt1SymbolInterleaverComponent.cpp.

References destroy().

```
81 {
82   destroy();
83 }
```

7.14.4 Member Function Documentation

7.14.4.1 template<typename T , size_t N> static T∗ iris::phy::Dvbt1SymbolInterleaverComponent::begin (T(&) *arr[N]*) [inline], [static], [private]

Useful templates.

Definition at line 130 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by Dvbt1SymbolInterleaverComponent().

```
130 { return &arr[0]; }
```

7.14.4.2 void iris::phy::Dvbt1SymbolInterleaverComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes,* std::map< std::string, int > & *outputTypes*) [virtual]

Calculate the output port types for the IRIS system.

The single output port must provide bytes.

Definition at line 98 of file Dvbt1SymbolInterleaverComponent.cpp.

```
101 {
102   outputTypes["output1"] = TypeInfo< uint8_t >::identifier;
103 }
```

7.14.4.3 void iris::phy::Dvbt1SymbolInterleaverComponent::destroy ( ) [private]

Destroy the component.

Definition at line 1095 of file Dvbt1SymbolInterleaverComponent.cpp.

References siRegister_.

Referenced by parameterHasChanged(), and ∼Dvbt1SymbolInterleaverComponent().

```
1096 {
1097   // clean
1098   delete [] siRegister_;
1099 }
```

7.14.4.4 template<typename T , size_t N> static T∗ iris::phy::Dvbt1SymbolInterleaverComponent::end (T(&) *arr[N]*) [inline], [static], [private]

Definition at line 132 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by Dvbt1SymbolInterleaverComponent().

```
132 { return &arr[0]+N; }
```

7.14.4.5 void iris::phy::Dvbt1SymbolInterleaverComponent::initialize ( ) [virtual]

Initialize the component.

Just calls setup().

Definition at line 108 of file Dvbt1SymbolInterleaverComponent.cpp.

References setup().

```
{
  setup();
}
```

7.14.4.6 void iris::phy::Dvbt1SymbolInterleaverComponent::parameterHasChanged (std::string *name*) [virtual]

Actions taken when the parameters change.

This block has one significant parameter

Definition at line 1073 of file Dvbt1SymbolInterleaverComponent.cpp.

References destroy(), and setup().

```
{
  if(name == "ofdmmode")
  {
    destroy();
    setup();
  }
}
```

7.14.4.7 void iris::phy::Dvbt1SymbolInterleaverComponent::process ( ) [virtual]

Main processing method.

Definition at line 1008 of file Dvbt1SymbolInterleaverComponent.cpp.

References debug_x, eo_, H_2K_, H_4K_, H_8K_, ofdmMode_x, siLength_, siOffset_, and siRegister_.

```
{
  // request input
  DataSet< uint8_t > *in = NULL;
  getInputDataSet("input1", in);

  // calculate sizes
  int insize = in ? (int) in->data.size() : 0;
  int outsize = siLength_ * ((insize + siOffset_) / siLength_);

  // request output
  DataSet< uint8_t >* out = NULL;
  getOutputDataSet("output1", out, outsize);

  // print debug info
  if(debug_x)
    LOG(LINFO) << "in/out: " << insize << "/" << outsize;

  // symbol by symbol
  for(ByteVecIt init = in->data.begin(), outit = out->data.begin(); init < in->data.end(); init++)
  {
    // copy to register
    siRegister_[siOffset_++] = *init;

    // trigger interleaving
    if(siOffset_ == siLength_)
    {
      // reset offset
      siOffset_ = 0;

      // actual interleaving matrix
      int *H = ofdmMode_x == 2048 ? H_2K_ : (ofdmMode_x == 4096 ? H_4K_ : H_8K_);

      // copy
      switch(eo_++ & 0x01)
      {
        case 0:
          // even numbered symbol
          for(int s = 0; s < siLength_; s++)
            outit[H[s]] = siRegister_[s];
          break;
        case 1:
          // odd numbered symbol
          for(int s = 0; s < siLength_; s++)
            outit[s] = siRegister_[H[s]];
          break;
      }

      // advance output
      outit += siLength_;
    }
  }

  // Copy the timestamp and sample rate for the DataSets
  out->timeStamp = in->timeStamp;
  out->sampleRate = in->sampleRate;

  // release input and output
  releaseInputDataSet("input1", in);
  releaseOutputDataSet("output1", out);
}
```

7.14.4.8 void iris::phy::Dvbt1SymbolInterleaverComponent::registerPorts ( ) [virtual]

Register the interleaver ports with the IRIS system.

This component has two inputs that accept symbols (some bits per byte) and one output that provides symbols (in one byte).

Definition at line 89 of file Dvbt1SymbolInterleaverComponent.cpp.

```
{
  registerInputPort("input1", TypeInfo< uint8_t >::identifier);
  registerOutputPort("output1", TypeInfo< uint8_t >::identifier);
}
```

7.14.4.9 void iris::phy::Dvbt1SymbolInterleaverComponent::setup ( ) [private]

Set up the register space and initialize.

Definition at line 1083 of file Dvbt1SymbolInterleaverComponent.cpp.

References eo_, ofdmMode_x, siLength_, siOffset_, and siRegister_.

Referenced by initialize(), and parameterHasChanged().

```
{
  // clean
  eo_ = 0;
  siOffset_ = 0;
  siLength_ = ofdmMode_x == 2048 ? 1512 : (ofdmMode_x ==  4096 ? 3024 : 6048);

  // get room
  siRegister_ = new uint8_t [siLength_];
}
```

7.14.5 Member Data Documentation

7.14.5.1 bool iris::phy::Dvbt1SymbolInterleaverComponent::debug_x [private]

Debug flag (default = false)

Definition at line 112 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by Dvbt1SymbolInterleaverComponent(), and process().

7.14.5.2 int iris::phy::Dvbt1SymbolInterleaverComponent::eo_ [private]

Even/odd numbered OFDM block.

Definition at line 124 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process(), and setup().

7.14.5.3 int iris::phy::Dvbt1SymbolInterleaverComponent::H_2K_ [static], [private]

Interleaving addresses for 2K.

Definition at line 126 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process().

7.14.5.4 int iris::phy::Dvbt1SymbolInterleaverComponent::H_4K_ [static], [private]

Interleaving addresses for 4K.

Definition at line 126 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process().

7.14.5.5 int iris::phy::Dvbt1SymbolInterleaverComponent::H_8K_ [static], [private]

Interleaving addresses for 8K.

Definition at line 126 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process().

7.14.5.6 int iris::phy::Dvbt1SymbolInterleaverComponent::ofdmMode_x [private]

OFDM mode (default = 2048)

Definition at line 113 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by Dvbt1SymbolInterleaverComponent(), process(), and setup().

7.14.5.7 double iris::phy::Dvbt1SymbolInterleaverComponent::sampleRate_ [private]

Sample rate of current frame.

Definition at line 119 of file Dvbt1SymbolInterleaverComponent.h.

7.14.5.8 int iris::phy::Dvbt1SymbolInterleaverComponent::siLength_ [private]

Interleaving register length.

Definition at line 122 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process(), and setup().

7.14.5.9 int iris::phy::Dvbt1SymbolInterleaverComponent::siOffset_ [private]

Interleaving offset.

Definition at line 121 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by process(), and setup().

7.14.5.10 uint8_t∗ iris::phy::Dvbt1SymbolInterleaverComponent::siRegister_ [private]

Actual interleaving register.

Definition at line 123 of file Dvbt1SymbolInterleaverComponent.h.

Referenced by destroy(), process(), and setup().

7.14.5.11 double iris::phy::Dvbt1SymbolInterleaverComponent::timeStamp_ [private]

Timestamp of current frame.

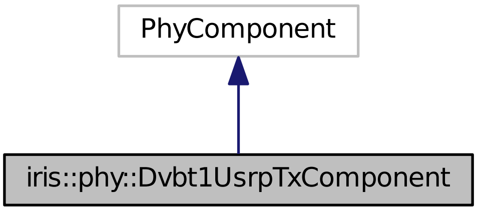
Definition at line 118 of file Dvbt1SymbolInterleaverComponent.h.

7.15 iris::phy::Dvbt1UsrpTxComponent Class Reference

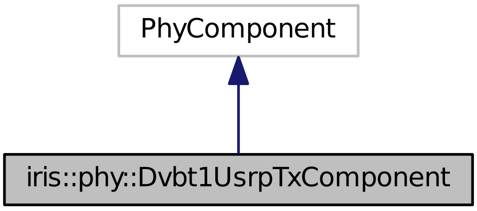
The Dvbt1UsrpTx component.

`#include <Dvbt1UsrpTxComponent.h>`

Inheritance diagram for iris::phy::Dvbt1UsrpTxComponent:



Collaboration diagram for iris::phy::Dvbt1UsrpTxComponent:



Public Member Functions

- Dvbt1UsrpTxComponent (std::string name)

- virtual ∼Dvbt1UsrpTxComponent ()
- virtual void calculateOutputTypes (std::map< std::string, int > &inputTypes, std::map< std::string, int > &outputTypes)
- virtual void registerPorts ()
- virtual void initialize ()

  Do any initialization required.
- virtual void process ()
- virtual void parameterHasChanged (std::string name)

  This gets called whenever a parameter is reconfigured.
- void usrpThreadProcedure ()

Private Attributes

- std::string args_x

  See `http://files.ettus.com/uhd_docs/manual/html/identification.html`.
- double rate_x

  Rate of outgoing samples.
- double frequency_x

  Tx frequency.
- double fixLoOffset_x

  Fix the local oscillator offset (defaults to 2∗rate)
- float gain_x

  Overall tx gain.
- std::string antenna_x

  Daughterboard antenna selection.
- std::string subDev_x

  Daughterboard subdevice specification.
- double bw_x

  Daughterboard IF filter bandwidth (Hz)
- std::string ref_x

  Reference waveform (internal, external, mimo)
- bool streaming_x

  Streaming or bursty traffic?
- std::string fmt_x

  Data format (fc64, fc32 or sc16)
- int bufferSize_x

  Size (in samples) of a single buffer.
- int numBuffers_x

  Number of buffers.
- ReadBuffer< std::complex
  < float > > ∗ inBuf_

  Convenience pointer to input buffer.
- uhd::usrp::multi_usrp::sptr usrp_

  The device.
- uhd::tx_streamer::sptr txStream_
- std::vector< std::vector
  < std::complex< float > > > bufs_
- std::vector< int > fulls_
- boost::condition_variable condW_
- boost::condition_variable condR_
- boost::mutex mutW_
- boost::mutex mutR_

- boost::mutex mut_
- int currentRead_
- int currentWrite_
- bool runUsrp_
- boost::thread ∗ usrpThread_

7.15.1 Detailed Description

The Dvbt1UsrpTx component.

A sink component which writes to a USRP transmitter using the Universal Hardware Driver (UHD). This component supports streaming data by default. This component is derived from the component used by the Iris modules. In addition to that, we use here additional threading and buffering to allow high data rate streams to be continuously and uninterruptedly transmitted from the input coming from other blocks to the real USRP device.

This block accepts in input complex float values.

There are several parameters that can be changed in the XML configuration file:

- *args*: by default set to "". This is a string that can be used to address one particular USRP device by specifying its IP address.
- *rate*: by default set to "1000000", it represents the sampling rate at which the digital samples are sent to the device.
- *frequency*: by default set to "2400000000", it is the frequency at which the BB signal, after digital to analog conversion, will be modulated.
- *gain*: by default set to "1", it is the gain of the final amplifier in the USRP TX chain.
- *streaming*: by default set to true, it states whether a continuous stream of samples has to be expected.
- *fixlooffset*: by default set to "0", this is the offset at which the analog oscillator will up-convert the BB signal, with respect to the specified frequency. This offset is recovered in digital by means of a digitally controlled oscillator implemented in the USRP FPGA.
- *antenna*: by default set to "", which means automatic selection. This parameter can be used to select one particular antenna, if more than one is present.
- *subdev*: by default set to "", which means automatic selection. This parameter allows selecting one particular subdevice inside of the device, if more than one is present.
- *bw*: by default set to "0" Hz, which means automatic selection. This parameter selects the bandwidth of the daughterboard IF filter, if present.
- *ref*: can be one of "internal", "external", "mimo", by default it is set to "internal". This parameter represents the type of clock reference signal used for the synchronization of all the device chips.
- *fmt*: can be one of "fc64" (double), "fc32" (float) or "sc16" (short), by default set to "fc32". This is the default sample precision used for sample representation.
- *numbuffers*: by default set to "4". A number of internal buffers is required to temporarily store the samples that are sent to the USRP. If buffers are not used, there could be moments during which the USRP (which runs inside of an asynchronous thread) is lacking input samples, thus degrading the quality of the emitted signal.
- *buffersize*: by default set to "1000000" samples, this is the number of samples contained in one of the buffers mentioned above.

Definition at line 99 of file Dvbt1UsrpTxComponent.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 iris::phy::Dvbt1UsrpTxComponent::Dvbt1UsrpTxComponent (std::string *name*)

Constructor

Call the constructor on PhyComponent and pass in all details about the component. Register all parameters and events in the constructor.

Parameters

name	The name assigned to this component when loaded

Definition at line 57 of file Dvbt1UsrpTxComponent.cpp.

References antenna_x, args_x, bufferSize_x, bw_x, fixLoOffset_x, fmt_x, frequency_x, gain_x, numBuffers_x, rate_x, ref_x, streaming_x, and subDev_x.

```
  : PhyComponent(name,
                "dvbt1usrptx",
                "UsrpTx with buffering and sustained TX rate",
                "Giuseppe Baruffa",
                "0.1")
  , currentRead_(0)
  , currentWrite_(0)
  , runUsrp_(true)
  , usrpThread_(NULL)
{
  /*
   * format:
   * registerParameter(name,
   *                   description,
   *                   default value,
   *                   dynamic?,
   *                   parameter,
   *                   allowed values);
   */
  registerParameter("args",
                    "A delimited string which may be used to specify a particular usrp",
                    "",
                    false,
                    args_x);
  registerParameter("rate",
                    "The transmit rate",
                    "1000000",
                    true,
                    rate_x);
  registerParameter("frequency",
                    "The transmit frequency",
                    "2400000000",
                    true,
                    frequency_x);
  registerParameter("gain",
                    "The transmit gain",
                    "1",
                    true,
                    gain_x);
  registerParameter("streaming",
                    "Whether we're streaming data to tx",
                    "true",
                    true,
                    streaming_x);
  registerParameter("fixlooffset",
                    "Value to fix LO offset to in Hz",
                    "0",
                    false,
                    fixLoOffset_x);
  registerParameter("antenna",
                    "Daughterboard antenna selection",
                    "",
                    false,
                    antenna_x);
  registerParameter("subdev",
                    "Daughterboard subdevice specification",
                    "",
                    false,
                    subDev_x);
  registerParameter("bw",
                    "Daughterboard IF filter bandwidth (Hz)",
                    "0",
                    false,
                    bw_x);
  registerParameter("ref",
                    "Reference waveform (internal, external, mimo)",
                    "internal",
                    false,
                    ref_x);
  registerParameter("fmt",
                    "Data format (fc64, fc32 or sc16)",
                    "fc32",
                    false,
                    fmt_x);
  registerParameter("numbuffers",
                    "Number of buffers",
                    "4",
                    false,
                    numBuffers_x);
  registerParameter("buffersize",
                    "Size of a buffer (in samples)",
                    "1000000",
                    false,
                    bufferSize_x);
}
```

7.15.2.2 iris::phy::Dvbt1UsrpTxComponent::∼Dvbt1UsrpTxComponent ( ) [virtual]

Destructor

Send an EOB packet to stop the Usrp

Definition at line 148 of file Dvbt1UsrpTxComponent.cpp.

References runUsrp_, txStream_, and usrpThread_.

```
{
  // stop thread
  runUsrp_ = false;
  usrpThread_->join();
  delete usrpThread_;

  //Send a mini EOB packet
  uhd::tx_metadata_t md;
  md.start_of_burst = false;
  md.end_of_burst   = true;
  vector< complex<float> > v;
#if 1
  if(txStream_ != NULL)
  {
    txStream_->send(&v.front(), 0, md);
  }
#endif
}
```

7.15.3 Member Function Documentation

7.15.3.1 void iris::phy::Dvbt1UsrpTxComponent::calculateOutputTypes (std::map< std::string, int > & *inputTypes*, std::map< std::string, int > & *outputTypes*) [virtual]

Calculate output data types

Based on the input data types, tell the system what output data types will be provided.

Parameters

inputTypes	The data types of the inputs which will be passed to this component
outputTypes	The data types of the outputs which will be generated by this component

Definition at line 189 of file Dvbt1UsrpTxComponent.cpp.

```
{
  //No output types
}
```

7.15.3.2 void iris::phy::Dvbt1UsrpTxComponent::initialize ( ) [virtual]

Do any initialization required.

Definition at line 197 of file Dvbt1UsrpTxComponent.cpp.

References antenna_x, args_x, bufferSize_x, bufs_, bw_x, currentRead_, currentWrite_, fixLoOffset_x, fmt_x, frequency_x, fulls_, gain_x, inBuf_, numBuffers_x, rate_x, ref_x, runUsrp_, subDev_x, txStream_, usrp_, usrpThread_, and usrpThreadProcedure().

```
{
  //uhd::set_thread_priority_safe();

  //Set up the input DataBuffer
  inBuf_ = castToType< complex<float> >(inputBuffers.at(0));

  // prepare buffers
  bufs_.resize(numBuffers_x);
  fulls_.resize(numBuffers_x);
  bufs_.resize(numBuffers_x);
  for(int i = 0; i < numBuffers_x; i++)
  {
    bufs_[i].resize(bufferSize_x);
    fulls_[i] = 0;
  }
  currentRead_ = numBuffers_x - 1;
  currentWrite_ = 0;

  // the thread
  if(usrpThread_)
  {
    runUsrp_ = false;
    usrpThread_->join();
    delete usrpThread_;
  }
  runUsrp_ = true;
  usrpThread_ = new boost::thread(boost::bind(&Dvbt1UsrpTxComponent::usrpThreadProcedure, this));

#if 1
  //Set up the usrp
  try
  {
    //Create the device
    LOG(LINFO) << "Creating the usrp device with args: " << args_x;
    usrp_ = uhd::usrp::multi_usrp::make(args_x);
    //Lock mboard clocks
    usrp_->set_clock_source(ref_x);
    //always select the subdevice first, the channel mapping affects the other settings
    if (subDev_x!="")
      usrp_->set_tx_subdev_spec(subDev_x);
    LOG(LINFO) << "Using Device: " << usrp_->get_pp_string();

    //Set rate
    LOG(LINFO) << "Setting TX Rate: " << (rate_x/1e6) << "Msps...";
    usrp_->set_tx_rate(rate_x);
    LOG(LINFO) << "Actual TX Rate: " << (usrp_->get_tx_rate()/1e6) << "Msps...";

    //Set frequency
    LOG(LINFO) << "Setting TX Frequency: " << (frequency_x/1e6) << "MHz...";
    double lo_offset = 0;  //Set LO offset to zero by default
    if(fixLoOffset_x >= 0)
      lo_offset = fixLoOffset_x;
    usrp_->set_tx_freq(tune_request_t(frequency_x, lo_offset));
    LOG(LINFO) << "Actual TX Frequency: " << (usrp_->get_tx_freq()/1e6) << "MHz";
    LOG(LINFO) << "RX LO offset: " << (lo_offset/1e6) << "MHz...";

    //We can only set the time on usrp2 devices
    if(usrp_->get_mboard_name().find("usrp1") == string::npos)
    {
      LOG(LINFO) << "Setting device timestamp to 0...";
      usrp_->set_time_now(uhd::time_spec_t((double)0));
    }

    //set the rf gain
    gain_range_t range = usrp_->get_tx_gain_range();
    LOG(LINFO) << "Gain range: " << range.to_pp_string();
    LOG(LINFO) << "Setting TX Gain: " << gain_x << " dB...";
    usrp_->set_tx_gain(gain_x);
    LOG(LINFO) << "Actual TX Gain: " <<  usrp_->get_tx_gain() << " dB...";

    //set the IF filter bandwidth
    if(bw_x!=0)
    {
      LOG(LINFO) << "Setting TX Bandwidth: " << bw_x << " MHz...";
      usrp_->set_tx_bandwidth(bw_x);
      LOG(LINFO) << "Actual TX Bandwidth: " << usrp_->get_tx_bandwidth() << " MHz...";
    }

    //Set the antenna
    if(antenna_x!="")
      usrp_->set_tx_antenna(boost::to_upper_copy(antenna_x));
    LOG(LINFO) << "Using TX Antenna: " << usrp_->get_tx_antenna();

    boost::this_thread::sleep(boost::posix_time::seconds(1)); //allow for some setup time

    //Check Ref and LO Lock detect
    std::vector<std::string> sensor_names;
    sensor_names = usrp_->get_tx_sensor_names(0);
    if (std::find(sensor_names.begin(),
                  sensor_names.end(),
                  "lo_locked") != sensor_names.end())
    {
      uhd::sensor_value_t lo_locked = usrp_->get_tx_sensor("lo_locked",0);
      LOG(LINFO) << "Checking TX: " << lo_locked.to_pp_string() << " ...";
      if(!lo_locked.to_bool())
        throw IrisException("Failed to lock LO");
    }
    sensor_names = usrp_->get_mboard_sensor_names(0);
    if ((ref_x == "mimo") and (std::find(sensor_names.begin(),
                                         sensor_names.end(),
                                         "mimo_locked") != sensor_names.end()))
    {
      uhd::sensor_value_t mimo_locked = usrp_->get_mboard_sensor("mimo_locked",0);
      LOG(LINFO) << "Checking TX: " << mimo_locked.to_pp_string() << " ...";
      if(!mimo_locked.to_bool())
        throw IrisException("Failed to lock LO");
    }
    if ((ref_x == "external") and (std::find(sensor_names.begin(),
                                             sensor_names.end(),
                                             "ref_locked") != sensor_names.end()))
    {
      uhd::sensor_value_t ref_locked = usrp_->get_mboard_sensor("ref_locked",0);
      LOG(LINFO) << "Checking TX: " << ref_locked.to_pp_string() << " ...";
      if(!ref_locked.to_bool())
        throw IrisException("Failed to lock LO");
    }

    //create a transmit streamer
    uhd::stream_args_t stream_args(fmt_x);
    txStream_ = usrp_->get_tx_stream(stream_args);
  }
  catch(std::exception& e)
  {
    throw IrisException(e.what());
  }
#endif
}
```

7.15.3.3 void iris::phy::Dvbt1UsrpTxComponent::parameterHasChanged (std::string *name*) [virtual]

This gets called whenever a parameter is reconfigured.

Definition at line 476 of file Dvbt1UsrpTxComponent.cpp.

References fixLoOffset_x, frequency_x, gain_x, rate_x, and usrp_.

```
{
#if 1
  try
  {
    if(name == "frequency")
    {
      LOG(LINFO) << "Setting TX Frequency: " << (frequency_x/1e6) << "MHz...";
      double lo_offset = 2*rate_x;  //Set LO offset to twice signal rate by default
      if(fixLoOffset_x >= 0)
      {
        lo_offset = fixLoOffset_x;
      }
      usrp_->set_tx_freq(tune_request_t(frequency_x, lo_offset));
      LOG(LINFO) << "LOG TX Frequency: " << (usrp_->get_tx_freq()/1e6) << "MHz";
    }
    else if(name == "rate")
    {
      LOG(LINFO) << "Setting TX Rate: " << (rate_x/1e6) << "Msps...";
      usrp_->set_tx_rate(rate_x);
      LOG(LINFO) << "Actual TX Rate: " << (usrp_->get_tx_rate()/1e6) << "Msps...";
    }
    else if(name == "gain")
    {
      gain_range_t range = usrp_->get_tx_gain_range();
      LOG(LINFO) << "Gain range: " << range.to_pp_string();
      LOG(LINFO) << "Setting TX Gain: " << gain_x << " dB...";
      usrp_->set_tx_gain(gain_x);
      LOG(LINFO) << "Actual TX Gain: " <<  usrp_->get_tx_gain() << " dB...";
    }
  }
  catch(std::exception &e)
  {
    throw IrisException(e.what());
  }
#endif
}
```

7.15.3.4 void iris::phy::Dvbt1UsrpTxComponent::process ( ) [virtual]

The main work of the component is carried out here

Take a DataSet from the input buffer and send to the usrp

Definition at line 345 of file Dvbt1UsrpTxComponent.cpp.

References bufferSize_x, bufs_, condR_, condW_, currentRead_, currentWrite_, dbgprintf, DUMP_STATUS, fulls_, inBuf_, mut_, mutR_, and numBuffers_x.

```
{
  //Get a DataSet from the input DataBuffer
  DataSet< complex<float> >* readDataSet = NULL;
  inBuf_->getReadData(readDataSet);

  size_t insize = readDataSet->data.size();

  // check buffers
  int remSize = insize;
  while(remSize > 0)
  {
    bool have_to_wait = true;
    int availSize = bufferSize_x - fulls_[currentWrite_];
    if(availSize > 0)
    {
      // there is room in this buffer
      dbgprintf("filling W(%d)...\n", currentWrite_);
      DUMP_STATUS();
      int dasize = min(availSize, remSize);
      copy(&(readDataSet->data[insize - remSize]), &(readDataSet->data[insize - remSize + dasize]), &(bufs_[currentWrite_][fulls_[currentWrite_]]));
      fulls_[currentWrite_] += dasize;
      remSize -= dasize;
    }
    else
    {
      {
        // try to change buffer
        boost::lock_guard<boost::mutex> lock(mut_);
        int nextWrite_ = currentWrite_ == (numBuffers_x - 1) ? 0 : (currentWrite_ + 1);
        dbgprintf("looking for W(%d)...\n", nextWrite_);
        DUMP_STATUS();
        if(nextWrite_ != currentRead_)
        {
          // update buffer
          currentWrite_ = nextWrite_;
          // notify the reader that the writer has done something
          condW_.notify_one();
          dbgprintf("notified a write\n");
          have_to_wait = false;
        }
      }

      // wait for a free read buffer to write into
      if(have_to_wait)
      {
        dbgprintf("awaiting a read...\n");
        boost::unique_lock<boost::mutex> lockR(mutR_);
        condR_.wait(lockR);
        dbgprintf("awaited a read\n");
      }
    }
  }

  //Release the DataSet
  inBuf_->releaseReadData(readDataSet);
}
```

7.15.3.5 void iris::phy::Dvbt1UsrpTxComponent::registerPorts ( ) [virtual]

Register the ports of this component

Ports are registered by name with a vector of valid data types permitted on those ports. This example has one input port with a single valid data type - complex<float>.

Definition at line 173 of file Dvbt1UsrpTxComponent.cpp.

```
{
  //Register all ports
  vector<int> validTypes;
  validTypes.push_back(TypeInfo< complex<float> >::identifier);

  //format:        (name, vector of valid types)
  registerInputPort("input1", validTypes);
}
```

7.15.3.6 void iris::phy::Dvbt1UsrpTxComponent::usrpThreadProcedure ( )

Definition at line 404 of file Dvbt1UsrpTxComponent.cpp.

References bufferSize_x, bufs_, condR_, condW_, currentRead_, currentWrite_, dbgprintf, DUMP_STATUS, fulls_, mut_, mutW_, numBuffers_x, runUsrp_, and txStream_.

Referenced by initialize().

```
{
  uhd::tx_metadata_t md;
  md.start_of_burst = false;
  md.end_of_burst = false;
  md.has_time_spec = false;
  double max_waiting_time = 0.5;

    uhd::set_thread_priority_safe();

  // wait to avoid premature death
  {
    dbgprintf("awaiting a write...\n");
    boost::unique_lock<boost::mutex> lockW(mutW_);
    condW_.wait(lockW);
    dbgprintf("awaited a write\n");
  }

  while(runUsrp_)
  {
    DUMP_STATUS();

    // data available? send data
    size_t num_tx_samps = 0;
    while(fulls_[currentRead_])
    {
      dbgprintf("pouring R(%d)...\n", currentRead_);

      num_tx_samps = txStream_->send(
        &bufs_[currentRead_][num_tx_samps], bufferSize_x - num_tx_samps, md,
  max_waiting_time
      );
      fulls_[currentRead_] -= num_tx_samps;

      /*boost::this_thread::sleep(boost::posix_time::milliseconds(200));
      DUMP_STATUS();
      fulls_[currentRead_] -= 300000;
      if(fulls_[currentRead_]<0)
        fulls_[currentRead_] = 0;*/

      /*boost::this_thread::sleep(boost::posix_time::milliseconds(1));
      fulls_[currentRead_] = 0;*/
    }

    // go to next read
    bool have_to_wait = true;
    {
      boost::lock_guard<boost::mutex> lock(mut_);
      int nextRead_ = currentRead_ == (numBuffers_x - 1) ? 0 : (currentRead_ + 1);
      dbgprintf("advancing R(%d)...\n", nextRead_);
      DUMP_STATUS();
      if(nextRead_ != currentWrite_)
      {
        // change and notify the writer that the reader has done something
        currentRead_ = nextRead_;
        condR_.notify_one();
        dbgprintf("notified a read\n");
        have_to_wait = false;
      }
    }

    // wait the writer
    if(have_to_wait)
    {
      dbgprintf("awaiting a write\n");
      boost::unique_lock<boost::mutex> lockW(mutW_);
      condW_.wait(lockW);
      dbgprintf("awaited a write\n");
    }
  }
}
```

7.15.4 Member Data Documentation

7.15.4.1 std::string iris::phy::Dvbt1UsrpTxComponent::antenna_x [private]

Daughterboard antenna selection.

Definition at line 121 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.2 std::string iris::phy::Dvbt1UsrpTxComponent::args_x [private]

See http://files.ettus.com/uhd_docs/manual/html/identification.html.

Definition at line 116 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.3 int iris::phy::Dvbt1UsrpTxComponent::bufferSize_x [private]

Size (in samples) of a single buffer.

Definition at line 127 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), process(), and usrpThreadProcedure().

7.15.4.4 std::vector< std::vector< std::complex<float> > > iris::phy::Dvbt1UsrpTxComponent::bufs_ [private]

Definition at line 133 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), process(), and usrpThreadProcedure().

7.15.4.5 double iris::phy::Dvbt1UsrpTxComponent::bw_x [private]

Daughterboard IF filter bandwidth (Hz)

Definition at line 123 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.6 boost::condition_variable iris::phy::Dvbt1UsrpTxComponent::condR_ [private]

Definition at line 135 of file Dvbt1UsrpTxComponent.h.

Referenced by process(), and usrpThreadProcedure().

7.15.4.7 boost::condition_variable iris::phy::Dvbt1UsrpTxComponent::condW_ [private]

Definition at line 135 of file Dvbt1UsrpTxComponent.h.

Referenced by process(), and usrpThreadProcedure().

7.15.4.8 int iris::phy::Dvbt1UsrpTxComponent::currentRead_ [private]

Definition at line 138 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), process(), and usrpThreadProcedure().

7.15.4.9 int iris::phy::Dvbt1UsrpTxComponent::currentWrite_ [private]

Definition at line 138 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), process(), and usrpThreadProcedure().

7.15.4.10 double iris::phy::Dvbt1UsrpTxComponent::fixLoOffset_x [private]

Fix the local oscillator offset (defaults to 2∗rate)

Definition at line 119 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), and parameterHasChanged().

7.15.4.11 std::string iris::phy::Dvbt1UsrpTxComponent::fmt_x [private]

Data format (fc64, fc32 or sc16)

Definition at line 126 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.12 double iris::phy::Dvbt1UsrpTxComponent::frequency_x [private]

Tx frequency.

Definition at line 118 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), and parameterHasChanged().

7.15.4.13 std::vector<int> iris::phy::Dvbt1UsrpTxComponent::fulls_ [private]

Definition at line 134 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), process(), and usrpThreadProcedure().

7.15.4.14 float iris::phy::Dvbt1UsrpTxComponent::gain_x [private]

Overall tx gain.

Definition at line 120 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), and parameterHasChanged().

7.15.4.15 ReadBuffer< std::complex<float> >* iris::phy::Dvbt1UsrpTxComponent::inBuf_ [private]

Convenience pointer to input buffer.

Definition at line 130 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), and process().

7.15.4.16 boost::mutex iris::phy::Dvbt1UsrpTxComponent::mut_ [private]

Definition at line 137 of file Dvbt1UsrpTxComponent.h.

Referenced by process(), and usrpThreadProcedure().

7.15.4.17 boost::mutex iris::phy::Dvbt1UsrpTxComponent::mutR_ [private]

Definition at line 136 of file Dvbt1UsrpTxComponent.h.

Referenced by process().

7.15.4.18 boost::mutex iris::phy::Dvbt1UsrpTxComponent::mutW_ [private]

Definition at line 136 of file Dvbt1UsrpTxComponent.h.

Referenced by usrpThreadProcedure().

7.15.4.19 int iris::phy::Dvbt1UsrpTxComponent::numBuffers_x [private]

Number of buffers.

Definition at line 128 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), process(), and usrpThreadProcedure().

7.15.4.20 double iris::phy::Dvbt1UsrpTxComponent::rate_x [private]

Rate of outgoing samples.

Definition at line 117 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), initialize(), and parameterHasChanged().

7.15.4.21 std::string iris::phy::Dvbt1UsrpTxComponent::ref_x [private]

Reference waveform (internal, external, mimo)

Definition at line 124 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.22 bool iris::phy::Dvbt1UsrpTxComponent::runUsrp_ [private]

Definition at line 139 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), usrpThreadProcedure(), and ∼Dvbt1UsrpTxComponent().

7.15.4.23 bool iris::phy::Dvbt1UsrpTxComponent::streaming_x [private]

Streaming or bursty traffic?

Definition at line 125 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent().

7.15.4.24 std::string iris::phy::Dvbt1UsrpTxComponent::subDev_x [private]

Daughterboard subdevice specification.

Definition at line 122 of file Dvbt1UsrpTxComponent.h.

Referenced by Dvbt1UsrpTxComponent(), and initialize().

7.15.4.25 uhd::tx_streamer::sptr iris::phy::Dvbt1UsrpTxComponent::txStream_ [private]

Definition at line 132 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), usrpThreadProcedure(), and ∼Dvbt1UsrpTxComponent().

7.15.4.26 uhd::usrp::multi_usrp::sptr iris::phy::Dvbt1UsrpTxComponent::usrp_ [private]

The device.

Definition at line 131 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), and parameterHasChanged().

7.15.4.27 boost::thread∗ iris::phy::Dvbt1UsrpTxComponent::usrpThread_ [private]

Definition at line 140 of file Dvbt1UsrpTxComponent.h.

Referenced by initialize(), and ∼Dvbt1UsrpTxComponent().

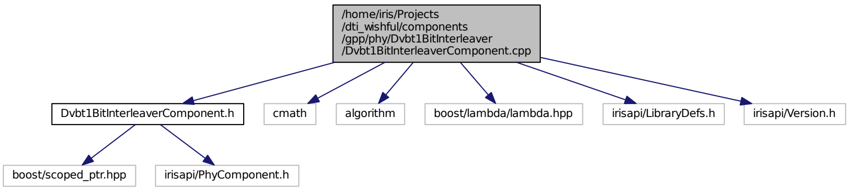
Chapter 8

File Documentation

8.1 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1BitInterleaver/Dvbt1BitInterleaverComponent.cpp File Reference

```
#include "Dvbt1BitInterleaverComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1BitInterleaverComponent.cpp:



/home/iris/Projects /dti_wishful/components /gpp/phy/Dvbt1BitInterleaver /Dvbt1BitInterleaverComponent.cpp
Dvbt1BitInterleaverComponent.h
cmath
algorithm
boost/lambda/lambda.hpp
irisapi/LibraryDefs.h
irisapi/Version.h
boost/scoped_ptr.hpp
irisapi/PhyComponent.h

Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1BitInterleaverComponent)

8.1.1 Detailed Description

Version

0.1

8.1.2 COPYRIGHT

8.1.3 LICENSE

8.1.4 DESCRIPTION

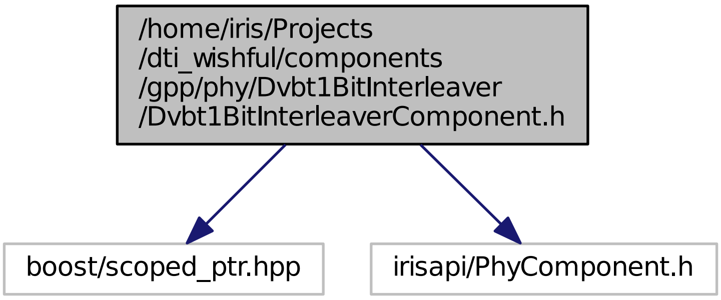
Implementation of the Dvbt1BitInterleaver component.

Definition in file Dvbt1BitInterleaverComponent.cpp.

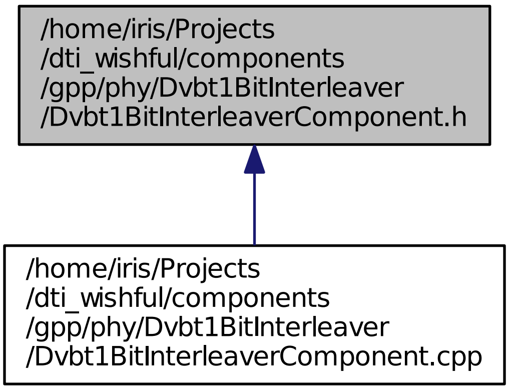
8.2 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1BitInterleaver/Dvbt1BitInterleaverComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1BitInterleaverComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1BitInterleaver
/Dvbt1BitInterleaverComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1BitInterleaver
/Dvbt1BitInterleaverComponent.cpp

Classes

- class iris::phy::Dvbt1BitInterleaverComponent

  A DVB-T1 bit interleaver component.

Namespaces

- iris
- iris::phy

8.2.1 Detailed Description

Version

0.1

8.2.2 COPYRIGHT

8.2.3 LICENSE

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.2.4 DESCRIPTION

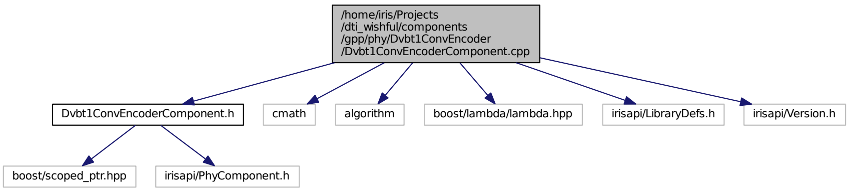
The Dvbt1BitInterleaver component.

Definition in file Dvbt1BitInterleaverComponent.h.

8.3 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1ConvEncoder/Dvbt1ConvEncoderComponent.cpp File Reference

```
#include "Dvbt1ConvEncoderComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1ConvEncoderComponent.cpp:



Namespaces

- iris
- iris::phy

Macros

- #define g1 0x4f

  First polynomial (bit-reversed)
- #define g2 0x6d

  Second polynomial (bit-reversed)

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ConvEncoderComponent)

8.3.1 Detailed Description

Version

0.1

8.3.2 COPYRIGHT

8.3.3 LICENSE

8.3.4 DESCRIPTION

Implementation of the Dvbt1ConvEncoder component.

Definition in file Dvbt1ConvEncoderComponent.cpp.

8.3.5 Macro Definition Documentation

8.3.5.1 #define g1 0x4f

First polynomial (bit-reversed)

Definition at line 146 of file Dvbt1ConvEncoderComponent.cpp.

Referenced by iris::phy::Dvbt1ConvEncoderComponent::process().

8.3.5.2 #define g2 0x6d

Second polynomial (bit-reversed)

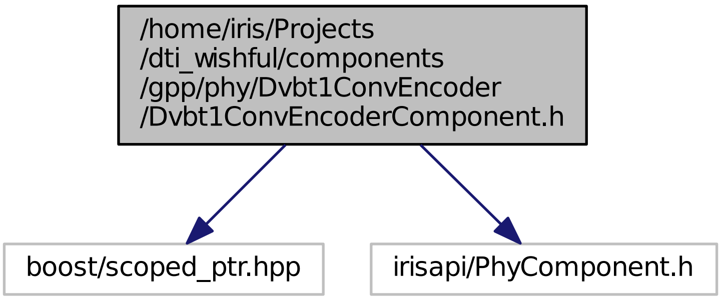
Definition at line 149 of file Dvbt1ConvEncoderComponent.cpp.

Referenced by iris::phy::Dvbt1ConvEncoderComponent::process().

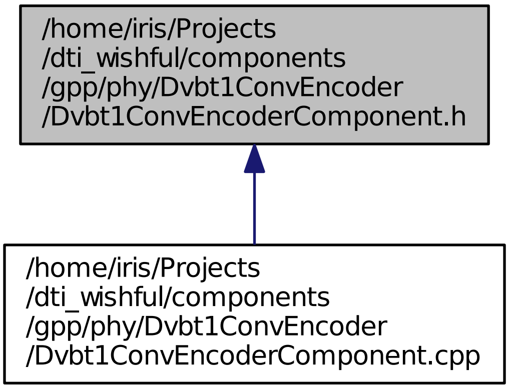
8.4 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1ConvEncoder/Dvbt1ConvEncoderComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1ConvEncoderComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1ConvEncoder
/Dvbt1ConvEncoderComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1ConvEncoder
/Dvbt1ConvEncoderComponent.cpp

Classes

- class iris::phy::Dvbt1ConvEncoderComponent

  A DVB-T1 convolutional encoder component.

Namespaces

- iris
- iris::phy

8.4.1 Detailed Description

Version

0.1

8.4.2 COPYRIGHT

8.4.3 LICENSE

8.4.4 DESCRIPTION

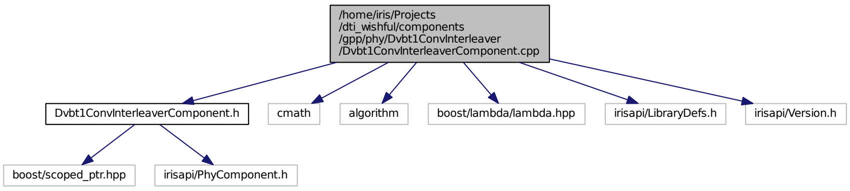
The Dvbt1ConvEncoder component.

Definition in file Dvbt1ConvEncoderComponent.h.

8.5 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1ConvInterleaver/Dvbt1ConvInterleaverComponent.cpp File Reference

```
#include "Dvbt1ConvInterleaverComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1ConvInterleaverComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ConvInterleaverComponent)

8.5.1 Detailed Description

Version

0.1

8.5.2 COPYRIGHT

8.5.3 LICENSE

8.5.4 DESCRIPTION

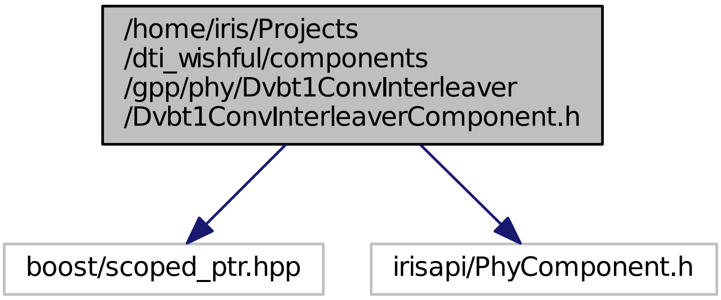
Implementation of the Dvbt1ConvInterleaver component.

Definition in file Dvbt1ConvInterleaverComponent.cpp.

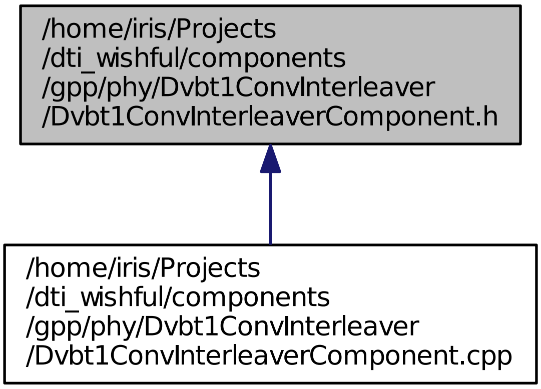
8.6 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1ConvInterleaver/Dvbt1ConvInterleaverComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1ConvInterleaverComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1ConvInterleaver
/Dvbt1ConvInterleaverComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1ConvInterleaver
/Dvbt1ConvInterleaverComponent.cpp

Classes

- class iris::phy::Dvbt1ConvInterleaverComponent

  A DVB-T1 convolutional interleaver component.

Namespaces

- iris
- iris::phy

8.6.1 Detailed Description

Version

0.1

8.6.2 COPYRIGHT

8.6.3 LICENSE

8.6.4 DESCRIPTION

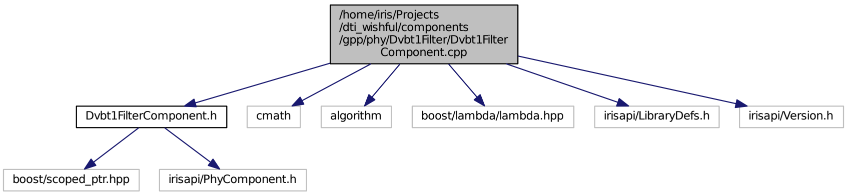
The Dvbt1ConvInterleaver component.

Definition in file Dvbt1ConvInterleaverComponent.h.

8.7 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Filter/Dvbt1FilterComponent.cpp File Reference

```
#include "Dvbt1FilterComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1FilterComponent.cpp:



Namespaces

- iris
- iris::phy

Macros

- #define MAX_FILTER_LENGTH 127

  Change this to suit your needs.
- #define MAX_FILTER_LENGTH_2 50001

  unused

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FilterComponent)

8.7.1 Detailed Description

Version

0.1

8.7.2 COPYRIGHT

8.7.3 LICENSE

8.7.4 DESCRIPTION

Implementation of the Dvbt1Filter component.

Definition in file Dvbt1FilterComponent.cpp.

8.7.5 Macro Definition Documentation

8.7.5.1 #define MAX_FILTER_LENGTH 127

Change this to suit your needs.

Definition at line 330 of file Dvbt1FilterComponent.cpp.

Referenced by iris::phy::Dvbt1FilterComponent::setup().

8.7.5.2 #define MAX_FILTER_LENGTH_2 50001

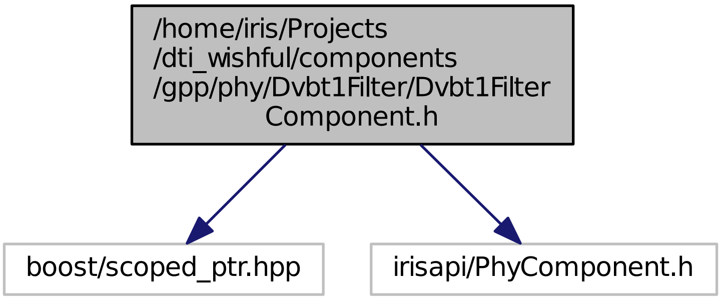
unused

Definition at line 333 of file Dvbt1FilterComponent.cpp.

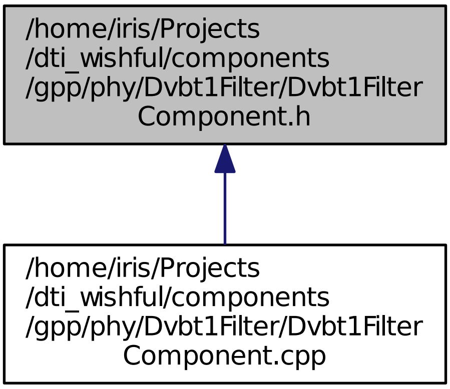
8.8 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Filter/Dvbt1FilterComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1FilterComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1FilterComponent

  A DVB-T1 filter component.

Namespaces

- iris
- iris::phy

8.8.1 Detailed Description

Version

0.1

8.8.2 COPYRIGHT

8.8.3 LICENSE

This file is part of the Iris Project.

8.8.4 DESCRIPTION

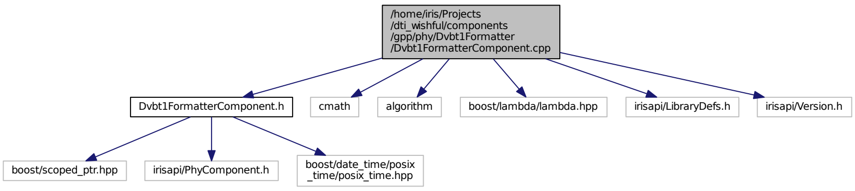
The Dvbt1Filter component.

Definition in file Dvbt1FilterComponent.h.

8.9 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Formatter/Dvbt1FormatterComponent.cpp File Reference

```
#include "Dvbt1FormatterComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1FormatterComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FormatterComponent)

8.9.1 Detailed Description

Version

0.1

8.9.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at `http://www.softwareradiosystems.com/iris/copyright.html`.

8.9.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at `http://www.gnu.org/licenses/`.

8.9.4 DESCRIPTION

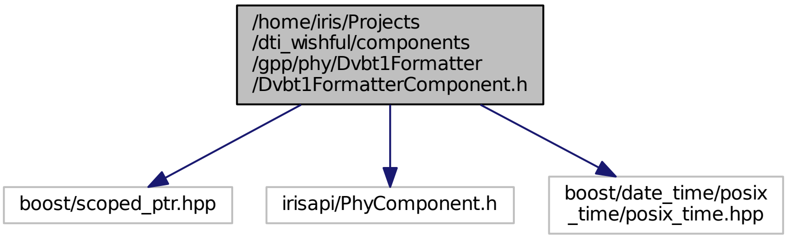
Implementation of the Dvbt1Formatter component.

Definition in file Dvbt1FormatterComponent.cpp.

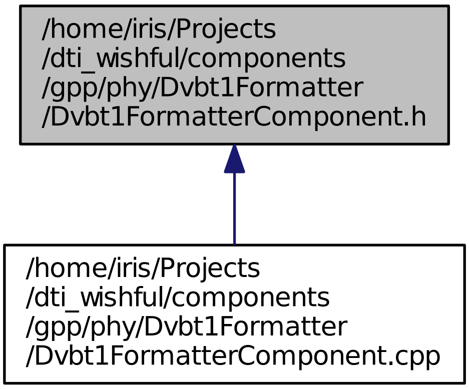
8.10 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Formatter/Dvbt1FormatterComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
#include <boost/date_time/posix_time/posix_time.hpp>
```

Include dependency graph for Dvbt1FormatterComponent.h:


/home/iris/Projects /dti_wishful/components /gpp/phy/Dvbt1Formatter /Dvbt1FormatterComponent.h
boost/scoped_ptr.hpp
irisapi/PhyComponent.h
boost/date_time/posix _time/posix_time.hpp

This graph shows which files directly or indirectly include this file:


/home/iris/Projects /dti_wishful/components /gpp/phy/Dvbt1Formatter /Dvbt1FormatterComponent.h
/home/iris/Projects /dti_wishful/components /gpp/phy/Dvbt1Formatter /Dvbt1FormatterComponent.cpp

Classes

- class iris::phy::Dvbt1FormatterComponent

  A DVB-T1 formatter component.

Namespaces

- iris
- iris::phy

8.10.1 Detailed Description

Version

0.1

8.10.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.10.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.10.4 DESCRIPTION

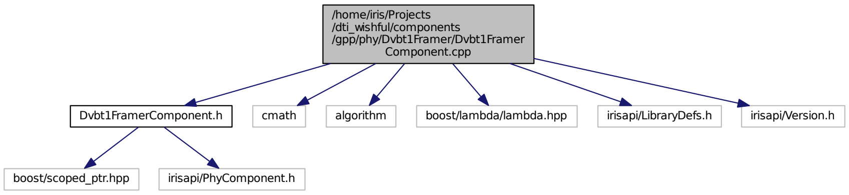
A formatter.

Definition in file Dvbt1FormatterComponent.h.

8.11 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Framer/Dvbt1FramerComponent.cpp File Reference

```
#include "Dvbt1FramerComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1FramerComponent.cpp:



Namespaces

- iris
- iris::phy

Macros

- #define T1_PIL_AMPL 1.333333333333F
- #define T1_TPS_AMPL 1.0F
- #define T1_N_BCH 127
- #define T1_K_BCH 113

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1FramerComponent)

8.11.1 Detailed Description

Version

0.1

8.11.2 COPYRIGHT

8.11.3 LICENSE

This file is part of the Iris Project.

8.11.4 DESCRIPTION

Implementation of the Dvbt1Framer component.

Definition in file Dvbt1FramerComponent.cpp.

8.11.5 Macro Definition Documentation

8.11.5.1 #define T1_K_BCH 113

Definition at line 381 of file Dvbt1FramerComponent.cpp.

Referenced by iris::phy::Dvbt1FramerComponent::t1_tps_generate().

8.11.5.2 #define T1_N_BCH 127

Definition at line 380 of file Dvbt1FramerComponent.cpp.

Referenced by iris::phy::Dvbt1FramerComponent::t1_tps_generate().

8.11.5.3 #define T1_PIL_AMPL 1.333333333333F

Definition at line 377 of file Dvbt1FramerComponent.cpp.

Referenced by iris::phy::Dvbt1FramerComponent::process().

8.11.5.4 #define T1_TPS_AMPL 1.0F

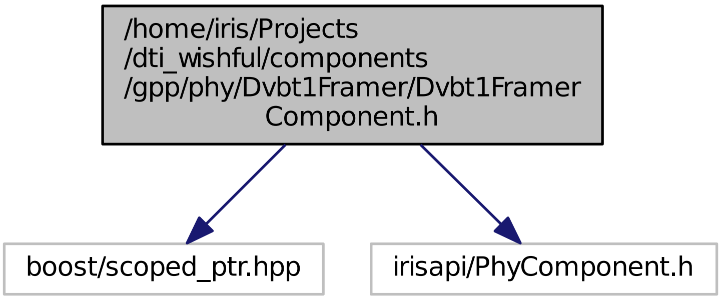
Definition at line 378 of file Dvbt1FramerComponent.cpp.

Referenced by iris::phy::Dvbt1FramerComponent::process().

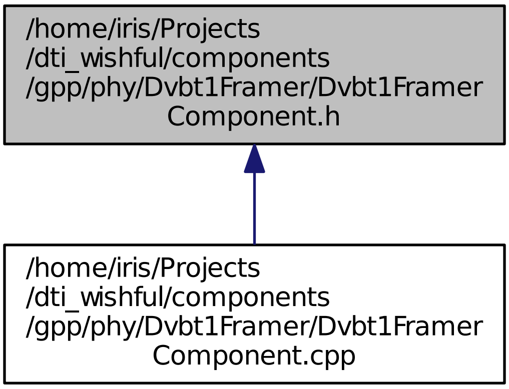
8.12 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Framer/Dvbt1FramerComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1FramerComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1FramerComponent

  A DVB-T1 framer component.

Namespaces

- iris
- iris::phy

Macros

- #define T1_BLOCKS_PER_FRAME 68
- #define T1_FRAMES_PER_SUPERFRAME 4

8.12.1 Detailed Description

Version

0.1

8.12.2 COPYRIGHT

8.12.3 LICENSE

This file is part of the Iris Project.

8.12.4 DESCRIPTION

The Dvbt1Framer component.

Definition in file Dvbt1FramerComponent.h.

8.12.5 Macro Definition Documentation

8.12.5.1 #define T1_BLOCKS_PER_FRAME 68

Definition at line 40 of file Dvbt1FramerComponent.h.

Referenced by iris::phy::Dvbt1FramerComponent::process().

8.12.5.2 #define T1_FRAMES_PER_SUPERFRAME 4

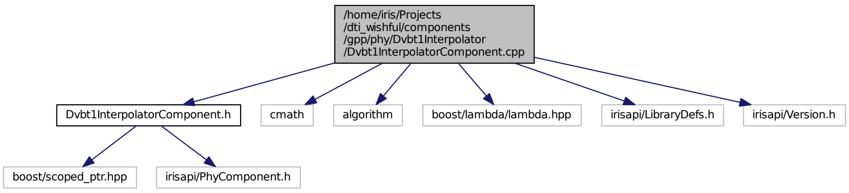
Definition at line 41 of file Dvbt1FramerComponent.h.

Referenced by iris::phy::Dvbt1FramerComponent::process().

8.13 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Interpolator/Dvbt1InterpolatorComponent.cpp File Reference

```
#include "Dvbt1InterpolatorComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1InterpolatorComponent.cpp:



Namespaces

- iris

- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1InterpolatorComponent)

8.13.1 Detailed Description

Version

0.1

8.13.2 COPYRIGHT

8.13.3 LICENSE

8.13.4 DESCRIPTION

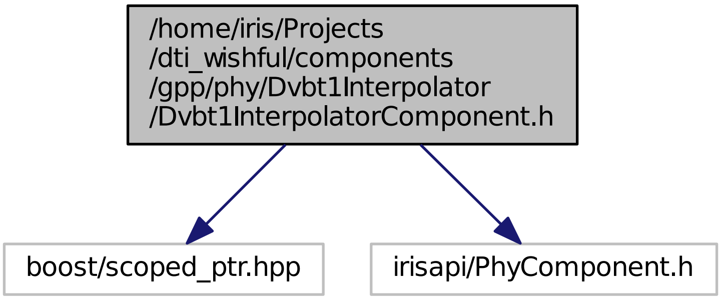
Implementation of the Dvbt1Interpolator component.

Definition in file Dvbt1InterpolatorComponent.cpp.

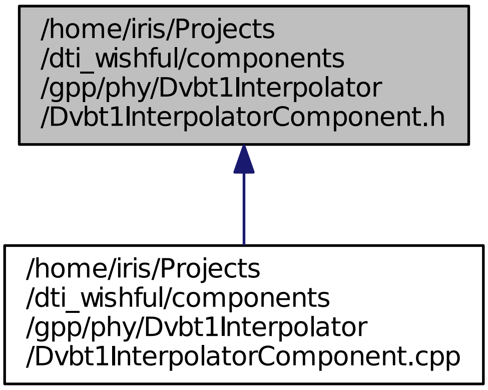
8.14 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Interpolator/Dvbt1InterpolatorComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1InterpolatorComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1Interpolator
/Dvbt1InterpolatorComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1Interpolator
/Dvbt1InterpolatorComponent.cpp

Classes

- class iris::phy::Dvbt1InterpolatorComponent

  A DVB-T1 interpolator component.

Namespaces

- iris
- iris::phy

Macros

- #define T1_RESAMPLE_ORDER 4

this defines the memory of the interpolator - keep low to have a good speed

8.14.1 Detailed Description

Version

0.1

8.14.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.14.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.14.4 DESCRIPTION

The Dvbt1Interpolator component.

Definition in file Dvbt1InterpolatorComponent.h.

8.14.5 Macro Definition Documentation

8.14.5.1 #define T1_RESAMPLE_ORDER 4

this defines the memory of the interpolator - keep low to have a good speed

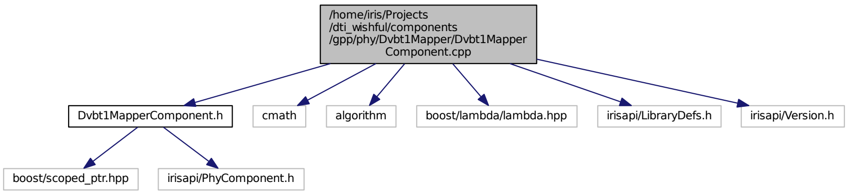
Definition at line 41 of file Dvbt1InterpolatorComponent.h.

Referenced by iris::phy::Dvbt1InterpolatorComponent::process(), iris::phy::Dvbt1InterpolatorComponent::setup(), and iris::phy::Dvbt1OfdmModComponent::setup().

8.15 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Mapper/Dvbt1MapperComponent.cpp File Reference

```
#include "Dvbt1MapperComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1MapperComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1MapperComponent)

8.15.1 Detailed Description

Version

0.1

8.15.2 COPYRIGHT

8.15.3 LICENSE

This file is part of the Iris Project.

8.15.4 DESCRIPTION

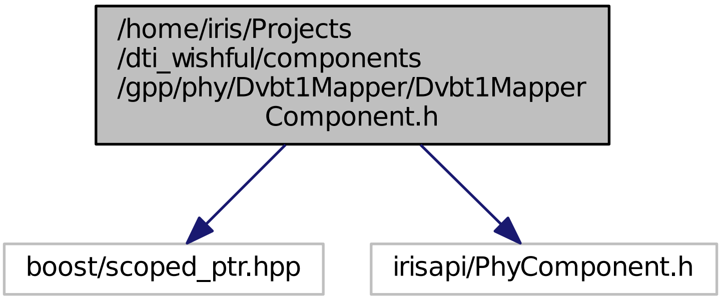
Implementation of the Dvbt1Mapper component.

Definition in file Dvbt1MapperComponent.cpp.

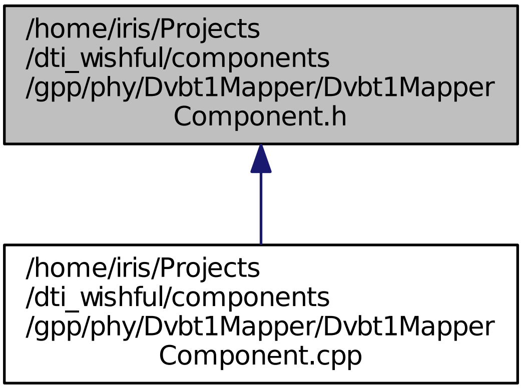
8.16 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Mapper/Dvbt1MapperComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1MapperComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1MapperComponent

  A DVB-T1 mapper component.

Namespaces

- iris

- iris::phy

8.16.1 Detailed Description

Version

0.1

8.16.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.16.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.16.4 DESCRIPTION

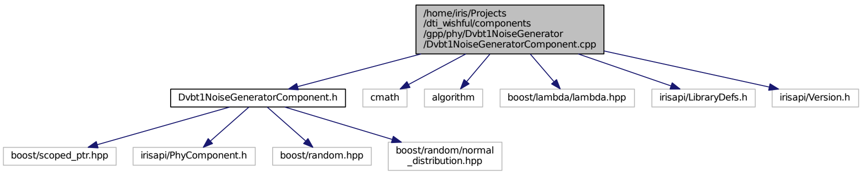
The Dvbt1Mapper component.

Definition in file Dvbt1MapperComponent.h.

8.17 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1NoiseGenerator/Dvbt1NoiseGeneratorComponent.cpp File Reference

```
#include "Dvbt1NoiseGeneratorComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1NoiseGeneratorComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1NoiseGeneratorComponent)

8.17.1 Detailed Description

Version

0.1

8.17.2 COPYRIGHT

8.17.3 LICENSE

8.17.4 DESCRIPTION

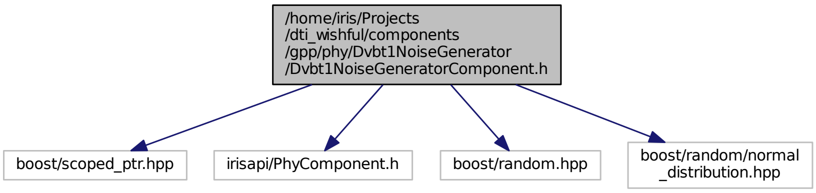
Implementation of the Dvbt1NoiseGenerator component.

Definition in file Dvbt1NoiseGeneratorComponent.cpp.

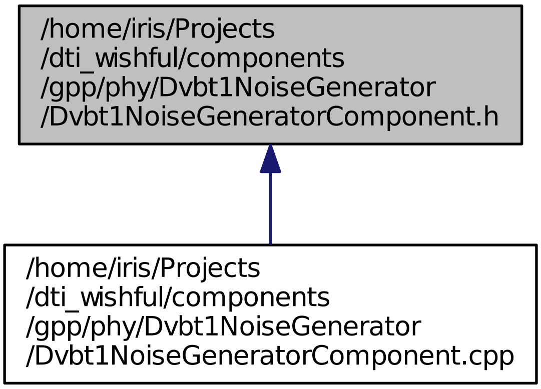
8.18 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1NoiseGenerator/Dvbt1NoiseGeneratorComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
#include <boost/random.hpp>
#include <boost/random/normal_distribution.hpp>
```

Include dependency graph for Dvbt1NoiseGeneratorComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1NoiseGenerator
/Dvbt1NoiseGeneratorComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1NoiseGenerator
/Dvbt1NoiseGeneratorComponent.cpp

Classes

- class iris::phy::Dvbt1NoiseGeneratorComponent

  A DVB-T1 noise generator.

Namespaces

- iris
- iris::phy

8.18.1 Detailed Description

Version

0.1

8.18.2 COPYRIGHT

8.18.3 LICENSE

8.18.4 DESCRIPTION

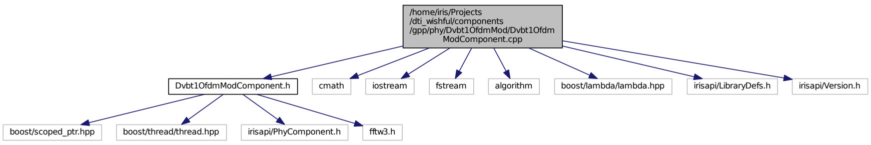
A DVB-T1 noise generator component.

Definition in file Dvbt1NoiseGeneratorComponent.h.

8.19 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1OfdmMod/Dvbt1OfdmModComponent.cpp File Reference

```
#include "Dvbt1OfdmModComponent.h"
#include <cmath>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1OfdmModComponent.cpp:



Namespaces

- iris
- iris::phy

Macros

- #define WAKEUPINTERVALMS 200

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1OfdmModComponent)

8.19.1 Detailed Description

Version

0.1

8.19.2 COPYRIGHT

8.19.3 LICENSE

8.19.4 DESCRIPTION

Implementation of the Dvbt1OfdmMod component.

Definition in file Dvbt1OfdmModComponent.cpp.

8.19.5 Macro Definition Documentation

8.19.5.1 #define WAKEUPINTERVALMS 200

Definition at line 402 of file Dvbt1OfdmModComponent.cpp.

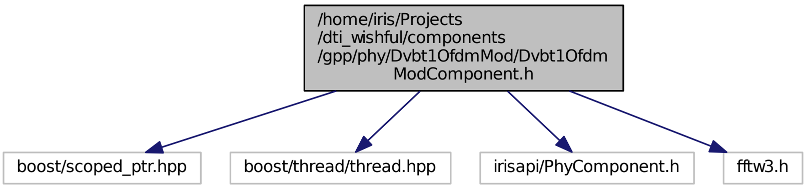
Referenced by iris::phy::Dvbt1OfdmModComponent::powerProcedure_().

8.20 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1OfdmMod/Dvbt1OfdmModComponent.h File Reference

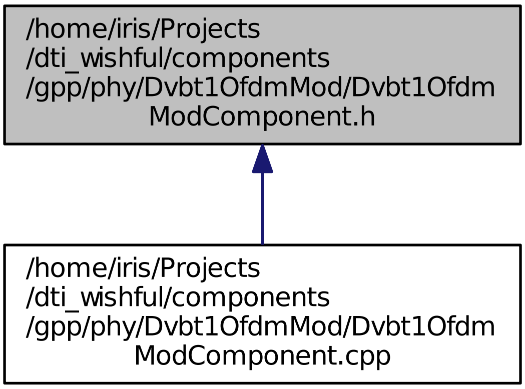
```
#include <boost/scoped_ptr.hpp>
```

```
#include <boost/thread/thread.hpp>
#include "irisapi/PhyComponent.h"
#include "fftw3.h"
```

Include dependency graph for Dvbt1OfdmModComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1OfdmModComponent

  A DVB-T1 OFDM modulator component.

Namespaces

- iris
- iris::phy

Macros

- #define T1_BLOCKS_PER_FRAME 68
- #define T1_FRAMES_PER_SUPERFRAME 4
- #define T1_RESAMPLE_ORDER 4

8.20.1 Detailed Description

Version

0.1

8.20.2 COPYRIGHT

8.20.3 LICENSE

8.20.4 DESCRIPTION

The Dvbt1OfdmMod component.

Definition in file Dvbt1OfdmModComponent.h.

8.20.5 Macro Definition Documentation

8.20.5.1 #define T1_BLOCKS_PER_FRAME 68

Definition at line 42 of file Dvbt1OfdmModComponent.h.

8.20.5.2 #define T1_FRAMES_PER_SUPERFRAME 4

Definition at line 43 of file Dvbt1OfdmModComponent.h.

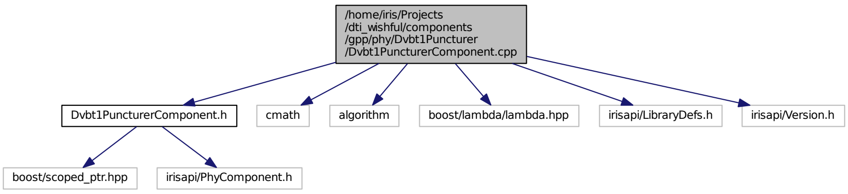
8.20.5.3 #define T1_RESAMPLE_ORDER 4

Definition at line 44 of file Dvbt1OfdmModComponent.h.

8.21 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Puncturer/Dvbt1PuncturerComponent.cpp File Reference

```
#include "Dvbt1PuncturerComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1PuncturerComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1PuncturerComponent)

8.21.1 Detailed Description

Version

0.1

8.21.2 COPYRIGHT

8.21.3 LICENSE

8.21.4 DESCRIPTION

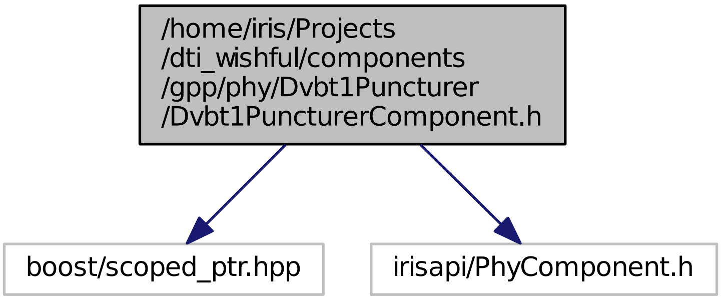
Implementation of the Dvbt1Puncturer component.

Definition in file Dvbt1PuncturerComponent.cpp.

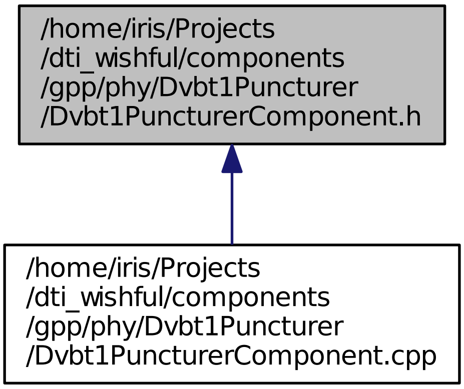
8.22 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Puncturer/Dvbt1PuncturerComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1PuncturerComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1PuncturerComponent

  A DVB-T1 puncturer component.

Namespaces

- iris
- iris::phy

8.22.1 Detailed Description

Version

0.1

8.22.2 COPYRIGHT

8.22.3 LICENSE

8.22.4 DESCRIPTION

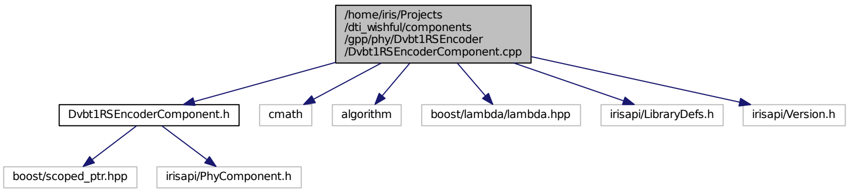
The Dvbt1Puncturer component.

Definition in file Dvbt1PuncturerComponent.h.

8.23 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1RSEncoder/Dvbt1RSEncoderComponent.cpp File Reference

```
#include "Dvbt1RSEncoderComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1RSEncoderComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1RSEncoderComponent)

8.23.1 Detailed Description

Version

0.1

8.23.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at `http://www.softwareradiosystems.com/iris/copyright.html`.

8.23.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at `http://www.gnu.org/licenses/`.

8.23.4 DESCRIPTION

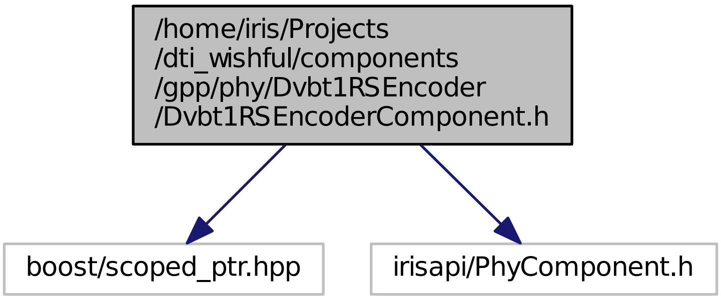
Implementation of the Dvbt1RSEncoder component.

Definition in file Dvbt1RSEncoderComponent.cpp.

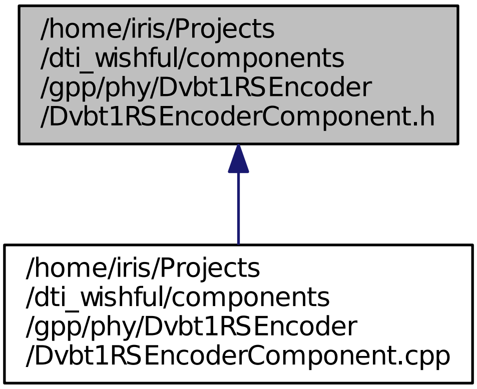
8.24 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1RSEncoder/Dvbt1RSEncoderComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1RSEncoderComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1RSEncoderComponent

  A DVB-T1 R-S Encoder component.

Namespaces

- iris

- iris::phy

Macros

- #define TS_PACKET_SIZE 188

  TS packet size.
- #define RS_PACKET_SIZE 204
- #define T1_MM 8

  R-S code over GF(2^8)
- #define T1_KK 239

  Nonshortened message size.
- #define T1_NN 255

  Nonshortened codeword size.
- #define T1_NN_KK 16

  Parity bytes size.
- #define T1_CLEAR(a, n)

  Clear an array from a point towards the beginning.
- #define T1_A0 (T1_NN)

  Placeholder for zero.

8.24.1 Detailed Description

Version

0.1

8.24.2 COPYRIGHT

8.24.3 LICENSE

8.24.4 DESCRIPTION

The Dvbt1RSEncoder component.

Definition in file Dvbt1RSEncoderComponent.h.

8.24.5 Macro Definition Documentation

8.24.5.1 #define RS_PACKET_SIZE 204

Definition at line 49 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::process().

8.24.5.2 #define T1_A0 (T1_NN)

Placeholder for zero.

Definition at line 70 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::packetEncode().

8.24.5.3 #define T1_CLEAR(*a*, *n*)

Value:

```
{\
    for(int ci=(n)-1;ci >=0;ci--)\
        (a)[ci] = 0;\
    }
```

Clear an array from a point towards the beginning.

Definition at line 64 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::packetEncode().

8.24.5.4 #define T1_KK 239

Nonshortened message size.

Definition at line 55 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::packetEncode(), and iris::phy::Dvbt1RSEncoderComponent::process().

8.24.5.5 #define T1_MM 8

R-S code over GF(2^8)

Definition at line 52 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::modnn().

8.24.5.6 #define T1_NN 255

Nonshortened codeword size.

Definition at line 58 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::modnn(), iris::phy::Dvbt1RSEncoderComponent::packetEncode(), iris::phy::Dvbt1RSEncoderComponent::process(), and iris::phy::Dvbt1RSEncoderComponent::setup().

8.24.5.7 #define T1_NN_KK 16

Parity bytes size.

Definition at line 61 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::packetEncode(), and iris::phy::Dvbt1RSEncoderComponent::process().

8.24.5.8 #define TS_PACKET_SIZE 188

TS packet size.

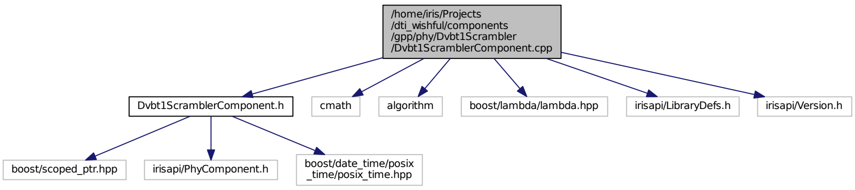
Definition at line 46 of file Dvbt1RSEncoderComponent.h.

Referenced by iris::phy::Dvbt1RSEncoderComponent::process().

8.25 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Scrambler/Dvbt1ScramblerComponent.cpp File Reference

```
#include "Dvbt1ScramblerComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1ScramblerComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1ScramblerComponent)

8.25.1 Detailed Description

Version

0.1

8.25.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.25.3 LICENSE

This file is part of the Iris Project.

8.25.4 DESCRIPTION

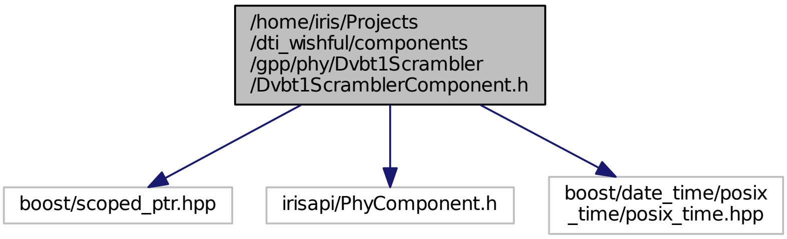
Implementation of the Dvbt1Scrambler component.

Definition in file Dvbt1ScramblerComponent.cpp.

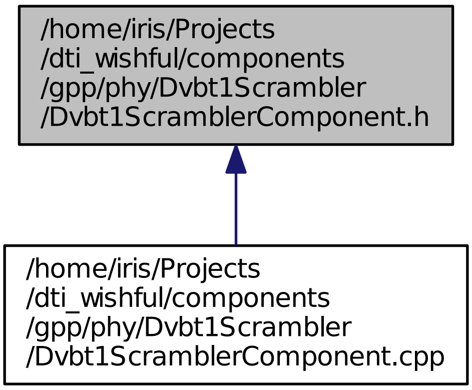
8.26 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1Scrambler/Dvbt1ScramblerComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
#include <boost/date_time/posix_time/posix_time.hpp>
```

Include dependency graph for Dvbt1ScramblerComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1ScramblerComponent

  A DVB-T energy dispersal component.

Namespaces

- iris
- iris::phy

8.26.1 Detailed Description

Version

0.1

8.26.2 COPYRIGHT

Copyright 2012-2016 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.26.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

8.26.4 DESCRIPTION

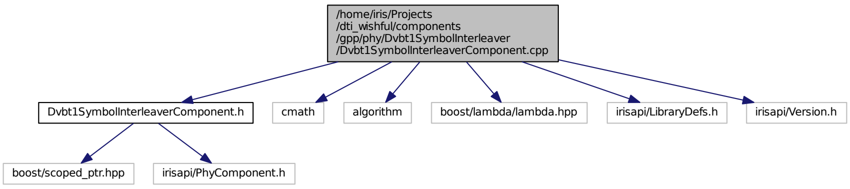
The Dvbt1Scrambler component.

Definition in file Dvbt1ScramblerComponent.h.

8.27 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1SymbolInterleaver/-Dvbt1SymbolInterleaverComponent.cpp File Reference

```
#include "Dvbt1SymbolInterleaverComponent.h"
#include <cmath>
#include <algorithm>
#include <boost/lambda/lambda.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1SymbolInterleaverComponent.cpp:



Namespaces

- iris
- iris::phy

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1SymbolInterleaverComponent)

8.27.1 Detailed Description

Version

0.1

8.27.2 COPYRIGHT

8.27.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at `http://www.gnu.org/licenses/`.

8.27.4 DESCRIPTION

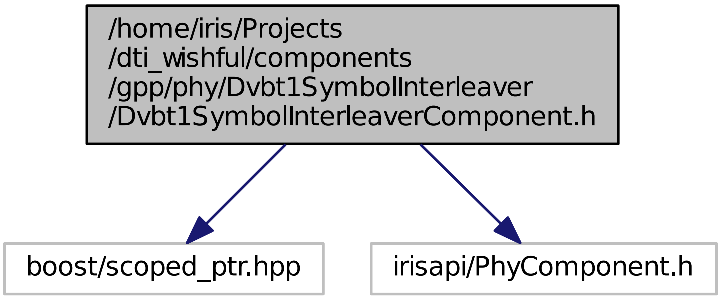
Implementation of the Dvbt1SymbolInterleaver component.

Definition in file Dvbt1SymbolInterleaverComponent.cpp.

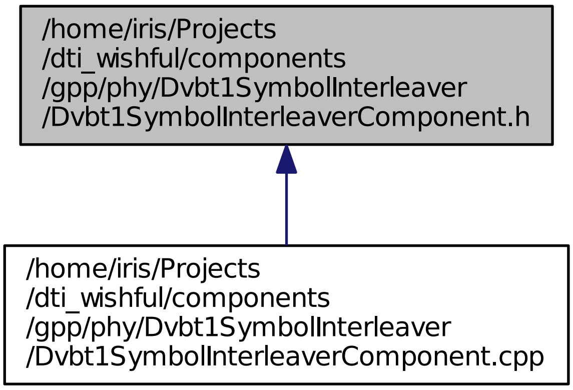
8.28 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1SymbolInterleaver/-Dvbt1SymbolInterleaverComponent.h File Reference

```
#include <boost/scoped_ptr.hpp>
#include "irisapi/PhyComponent.h"
```

Include dependency graph for Dvbt1SymbolInterleaverComponent.h:



This graph shows which files directly or indirectly include this file:


/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1SymbolInterleaver
/Dvbt1SymbolInterleaverComponent.h
/home/iris/Projects
/dti_wishful/components
/gpp/phy/Dvbt1SymbolInterleaver
/Dvbt1SymbolInterleaverComponent.cpp

Classes

- class iris::phy::Dvbt1SymbolInterleaverComponent

  A DVB-T1 symbol interleaver component.

Namespaces

- iris
- iris::phy

8.28.1 Detailed Description

Version

0.1

8.28.2 COPYRIGHT

8.28.3 LICENSE

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.28.4 DESCRIPTION

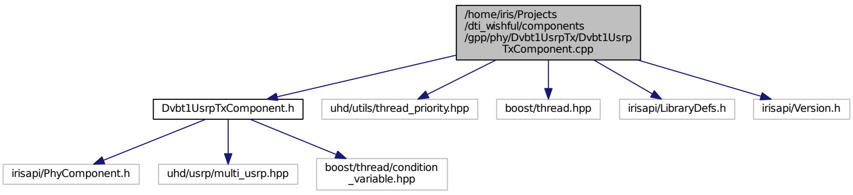
The Dvbt1SymbolInterleaver component.

Definition in file Dvbt1SymbolInterleaverComponent.h.

8.29 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1UsrpTx/Dvbt1UsrpTxComponent.cpp File Reference

```
#include "Dvbt1UsrpTxComponent.h"
#include <uhd/utils/thread_priority.hpp>
#include <boost/thread.hpp>
#include "irisapi/LibraryDefs.h"
#include "irisapi/Version.h"
```

Include dependency graph for Dvbt1UsrpTxComponent.cpp:



Namespaces

- iris
- iris::phy

Macros

- #define DUMP_STATUS() {}
- #define dbgprintf(...) {}

Functions

- iris::phy::IRIS_COMPONENT_EXPORTS (PhyComponent, Dvbt1UsrpTxComponent)

8.29.1 Detailed Description

Version

1.0

8.29.2 COPYRIGHT

8.29.3 LICENSE

Definition in file Dvbt1UsrpTxComponent.cpp.

8.29.4 Macro Definition Documentation

8.29.4.1 #define dbgprintf(...) {}

Definition at line 339 of file Dvbt1UsrpTxComponent.cpp.

Referenced by iris::phy::Dvbt1UsrpTxComponent::process(), and iris::phy::Dvbt1UsrpTxComponent::usrpThreadProcedure().

8.29.4.2 #define DUMP_STATUS( ) {}

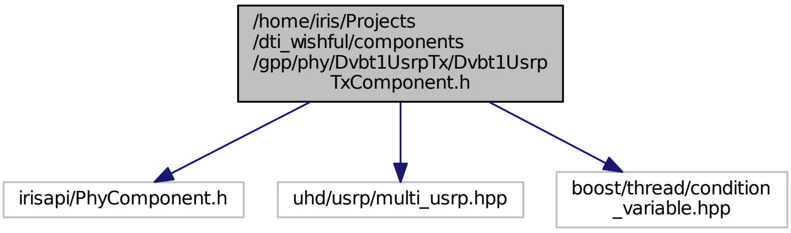
Definition at line 335 of file Dvbt1UsrpTxComponent.cpp.

Referenced by iris::phy::Dvbt1UsrpTxComponent::process(), and iris::phy::Dvbt1UsrpTxComponent::usrpThreadProcedure().

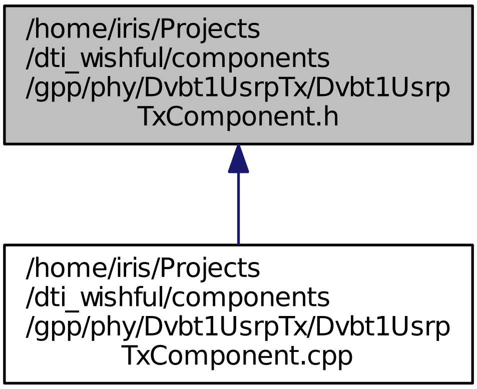
8.30 /home/iris/Projects/dti_wishful/components/gpp/phy/Dvbt1UsrpTx/Dvbt1UsrpTxComponent.h File Reference

```
#include "irisapi/PhyComponent.h"
#include <uhd/usrp/multi_usrp.hpp>
#include <boost/thread/condition_variable.hpp>
```

Include dependency graph for Dvbt1UsrpTxComponent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class iris::phy::Dvbt1UsrpTxComponent

  The Dvbt1UsrpTx component.

Namespaces

- iris
- iris::phy

8.30.1 Detailed Description

Version

1.0

8.30.2 COPYRIGHT

Copyright 2012-2013 The Iris Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://www.softwareradiosystems.com/iris/copyright.html.

8.30.3 LICENSE

This file is part of the Iris Project.

Iris is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Iris is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License can be found in the LICENSE file in the top-level directory of this distribution and at http://www.gnu.org/licenses/.

8.30.4 DESCRIPTION

The Dvbt1UsrpTx component.

Definition in file Dvbt1UsrpTxComponent.h.

8.31 Main_Page.txt File Reference

Chapter 9

Example Documentation

9.1 dvbt1chain_ofdmmod_filter_spectrum.xml

This is an example of how to use the DVB-TX-IRIS modules to show the spectrum of the generated DVB-T signal.

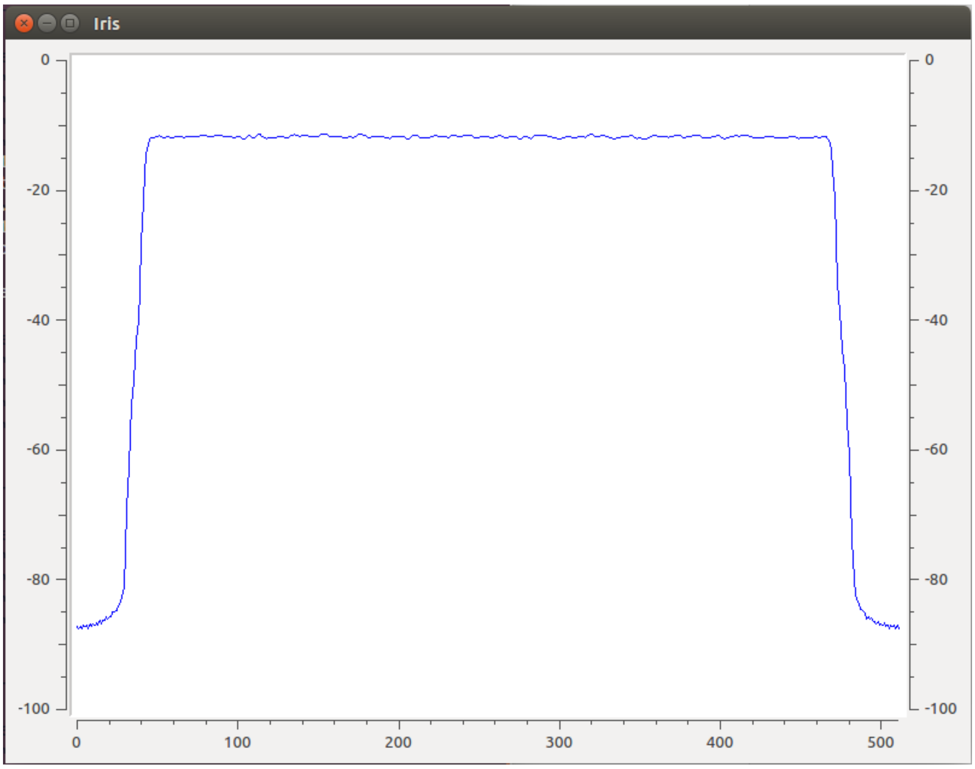


Figure 9.1: Spectrum of the generated DVB-T signal.

The transmission chain has been split among several physical engines (i.e., separate threads), so as to have the maximum parallelized performance.

```
<?xml version="1.0" encoding="utf-8" ?>

<softwareradio name="Radio1">

  <controller class="spectrogramdisplay">
    <parameter name="spectrogramcomponent" value="spectrogram1"/>
  </controller>

  <engine name="phyengine1" class="phyengine">
```

```
<component name="filerawreader1" class="filerawreader">
  <parameter name="filename" value="mux4800000.ts"/>
  <parameter name="blocksize" value="4096"/>
  <parameter name="datatype" value="uint8_t"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1scrambler1" class="dvbt1scrambler">
  <parameter name="debug" value="false"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1rsencoder1" class="dvbt1rsencoder">
  <parameter name="debug" value="false"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1convinterleaver1" class="dvbt1convinterleaver">
  <parameter name="debug" value="false"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1convencoder1" class="dvbt1convencoder">
  <parameter name="debug" value="false"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1puncturer1" class="dvbt1puncturer">
  <parameter name="debug" value="false"/>
  <parameter name="coderate" value="34"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1bitinterleaver1" class="dvbt1bitinterleaver">
  <parameter name="debug" value="false"/>
  <parameter name="qammapping" value="64"/>
  <parameter name="hyerarchymode" value="0"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1symbolinterleaver1" class="dvbt1symbolinterleaver">
  <parameter name="debug" value="false"/>
  <parameter name="ofdmmode" value="2048"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1mapper1" class="dvbt1mapper">
  <parameter name="debug" value="false"/>
  <parameter name="qammapping" value="64"/>
  <parameter name="hyerarchymode" value="0"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1framer1" class="dvbt1framer">
  <parameter name="debug" value="false"/>
  <parameter name="ofdmmode" value="2048"/>
  <parameter name="qammapping" value="64"/>
  <parameter name="hyerarchymode" value="0"/>
  <parameter name="cellid" value="-1"/>
  <parameter name="hpcoderate" value="34"/>
  <parameter name="indepthinterleaver" value="false"/>
  <parameter name="deltamode" value="32"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1ofdmmod1" class="dvbt1ofdmmod">
  <parameter name="debug" value="false"/>
  <parameter name="ofdmmode" value="2048"/>
  <parameter name="deltamode" value="32"/>
  <parameter name="outpower" value="50"/>
  <port name="input1" class="input"/>
  <port name="output1" class="output"/>
</component>

<component name="dvbt1filter1" class="dvbt1filter">
  <parameter name="debug" value="false"/>
  <parameter name="samplerate" value="9142857.143"/>
```

```
      <parameter name="stopband" value="4000000"/>
      <parameter name="attenuation" value="40"/>
      <port name="input1" class="input"/>
      <port name="output1" class="output"/>
    </component>

    <component name="spectrogram1" class="spectrogram">
      <parameter name="isprobe" value="true"/>
      <parameter name="issink" value="true"/>
      <parameter name="nwindows" value="512"/>
      <parameter name="nfft" value="512"/>
     <port name="input1" class="input"/>
      <port name="output1" class="output"/>
    </component>

  </engine>

  <link source="filerawreader1.output1" sink="dvbt1scrambler1.input1" />
  <link source="dvbt1scrambler1.output1" sink="dvbt1rsencoder1.input1" />
  <link source="dvbt1rsencoder1.output1" sink="dvbt1convinterleaver1.input1" />
  <link source="dvbt1convinterleaver1.output1" sink="dvbt1convencoder1.input1" />
  <link source="dvbt1convencoder1.output1" sink="dvbt1puncturer1.input1" />
  <link source="dvbt1puncturer1.output1" sink="dvbt1bitinterleaver1.input1" />
  <link source="dvbt1bitinterleaver1.output1" sink="dvbt1symbolinterleaver1.input1" />
  <link source="dvbt1symbolinterleaver1.output1" sink="dvbt1mapper1.input1" />
  <link source="dvbt1mapper1.output1" sink="dvbt1framer1.input1" />
  <link source="dvbt1framer1.output1" sink="dvbt1ofdmmod1.input1" />
  <link source="dvbt1ofdmmod1.output1" sink="dvbt1filter1.input1" />
  <link source="dvbt1filter1.output1" sink="spectrogram1.input1" />

</softwareradio>
```

Index