



Product Specification and User Guide
Distributed Half-Band Filter

PART NUMBER
DHBF-63-80-95-XV2-14-16-L



TABLE OF CONTENTS

Overview 3
Core version 3
Deliverables 3
General description 3
DHBF input format 5
Internal precision and scaling 5
DHBF output formats 7
Core input synchronisation 8
Core output synchronisation 8
Pipeline Latency 8
Core interface description 8
Place and Route report showing silicon usage 10
Timing 11
Simulated Power Dissipation Report 11
Verification 12
Tools 12
Delivered file directory structure 12
Using the core 13
Using the VHDL test bench 14
Running the test bench for RTL simulation 14
Running the test bench for VITAL simulation 15
Using Matlab Models 16

OVERVIEW

This document describes the RF Engines Ltd (RFEL) Distributed Half-Band Filter (DHBF) core. The DHBF is a 'down-convert and decimate' filter, primarily intended for use as the first stage in very high sample rate DSP systems, converting real data to complex. The core processes continuous real data, with no gaps. The core is provided in EDIF netlist form as a component.

CORE VERSION

1.0

DELIVERABLES

Supplied Item	Description
Design	EDIF netlist
Constraints File	UCF (User Constraints File)
Instantiation Template	VHDL
Verification	VHDL test bench including ModelSim script and test data files. Compiled RTL VHDL Model. Bit-true Matlab model and scripts. Placement reports.

Table 1: Items provided with each core

GENERAL DESCRIPTION

The basic DHBF architecture is shown in Figure 1 and performs the following three functions:

- Complex $f_s/4$ down-conversion from IF to base-band for Odd and even Nyquist regions
- Half-band filtering to remove aliases
- Decimation by two

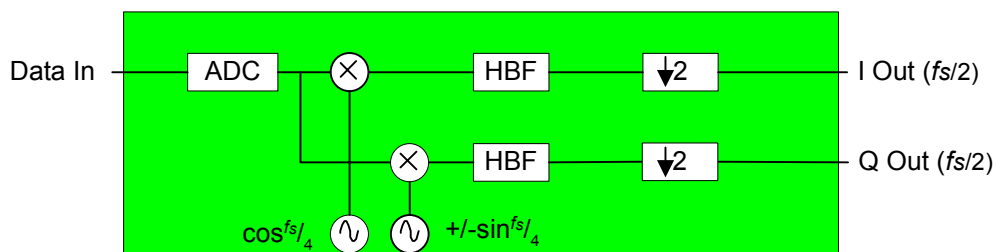


Figure 1: Basic DHBF

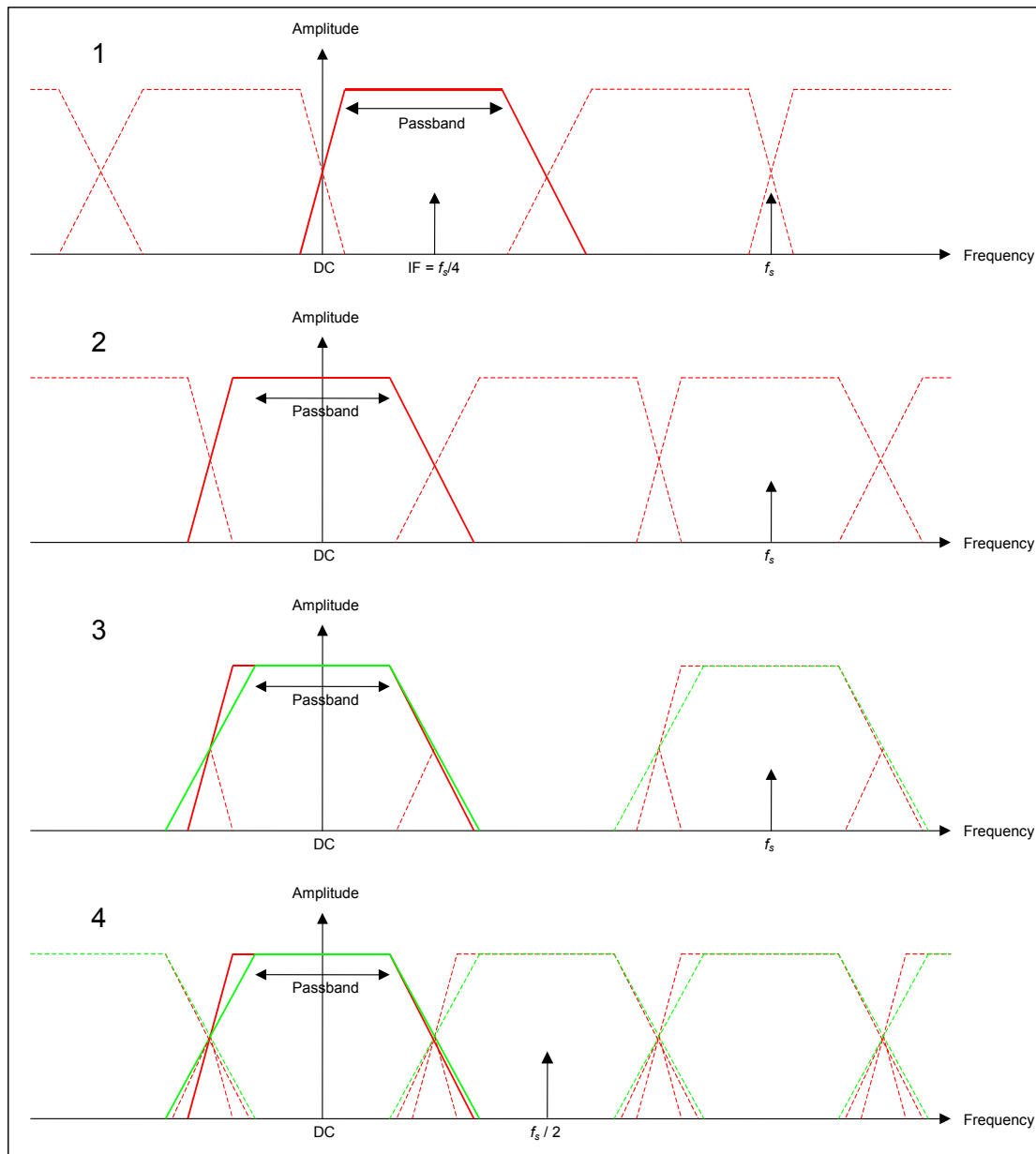


Figure 2: DHBF Process Frequency Plots

The processes shown in Figure 2 are described as follows:

The analogue signal is passed through a low-pass anti-alias filter, and then sampled at a frequency of f_s . A typical cascaded frequency response of an analogue down-conversion chain (low-pass anti-alias filter cascaded with band-pass anti-alias filters that have been used in the down-conversion process) is shown by the solid red line in plot 1. A negative frequency image of the analogue anti-alias filter is shown as a dashed red line, reflected about DC in the frequency domain. Further image pairs are repeated at intervals of $\pm f_s$ across the entire spectrum, and those present in the plotted frequency range are also shown in dashed red on plot 1.

The sampled real data stream is digitally mixed with a complex frequency of $-f_s/4$ to convert the IF at $f_s/4$ down to complex base-band. This mixing process does not add any further images, but just shifts the entire spectrum down by a frequency of $f_s/4$ as shown in plot 2.

Note. The frequency plan shows the IF centred in the first Nyquist region. Any other Nyquist region could be used, providing the analogue front-end and digitiser performance is acceptable for the input frequency band. If, for example, the second Nyquist region were used, a mixing frequency of $-3f_s/4$ ($= f_s/4$) would convert the IF to base-band, and the analogue anti-alias filter would be a passband filter centred on $3f_s/4$.

A digital half-band filter is applied to the complex base-band signal to eliminate the negative frequency sampling aliases as shown. The half-band filter is shown as the symmetrical green form overlaying the red analogue anti-alias filter in plot 3. The main properties of the half-band filter that make it an attractive choice for this low-pass stage are extremely low pass-band ripple, linear phase and alternating zero value coefficients that enable significant hardware optimisations. Filtering of the negative frequency sampling aliases is necessary to allow decimation.

Finally, the filtered complex base-band signal is decimated by two to produce a critically sampled complex data stream with a complex sample rate of $f_s/2$. The effect of decimation in the frequency domain is to effectively overlap the spectrum with an $f_s/2$ shifted version of itself, producing the dashed aliases shown in plot 4.

As the frequency plots show, the passband of the final base-band signal is alias-free, provided that the analogue anti-alias and digital half-band filters are correctly designed.

DHBF INPUT FORMAT

The DHBF core is designed to process continuous real data, input at the core clock rate. The format of the data input is signed 2's-complement. The signal 'start_p' is used to mark the input sample that corresponds to time zero, as shown in Figure 5. This input sets the phase of the mixing process within the core.

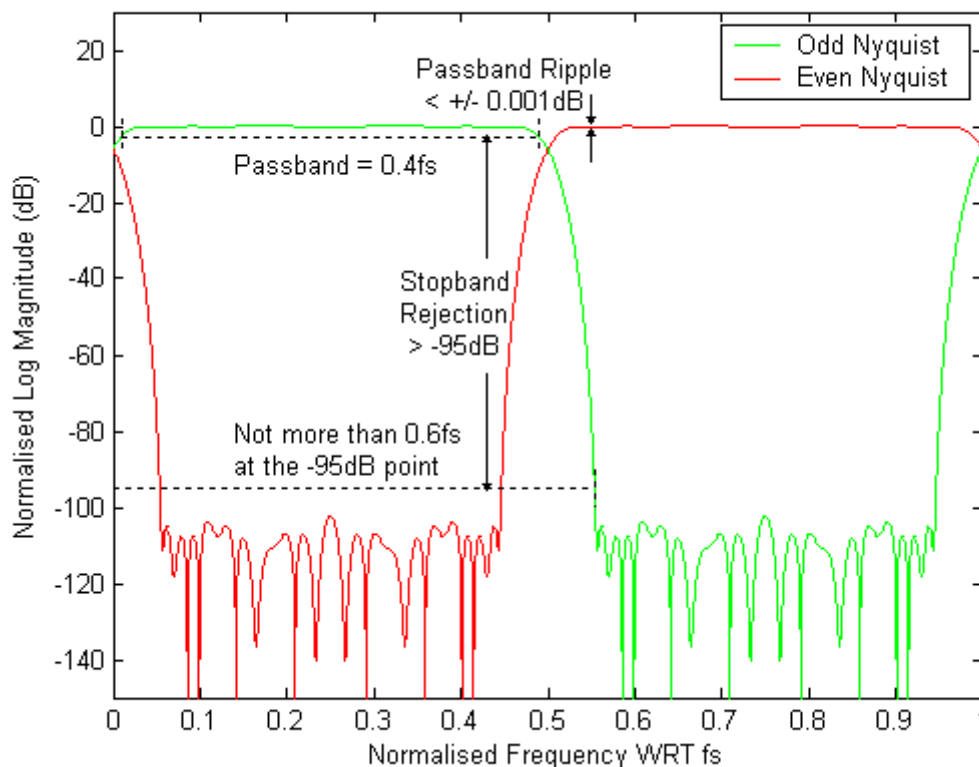
Note. The input data rate into the core is normally the A to D sample rate, f_s . The DHBF core must be clocked at this rate of f_s .

INTERNAL PRECISION AND SCALING

The DHBF filter parameters are given in Table 2 and the frequency response is shown in Figure 3.

Parameter	Specification
IF Input Frequency	Centre of odd or even Nyquist regions (selectable) i.e. IF of 175MHz, f_s of 100MHz gives 4 th , (even), Nyquist.
Input Precision	14 Bits (real)
Output Phase	0°
Output Rounding	Symmetrical
No of Taps	63

Filter coefficient values	[-5, 0, 22, 0, -67, 0, 164, 0, -353, 0, 688, 0, -1244, 0, 2120, 0, -3443, 0, 5386, 0, -8205, 0, 12335, 0, -18665, 0, 29478, 0, -53204, 0, 166063, 262144, 166063, 0, -53204, 0, 29478, 0, -18665, 0, 12335, 0, -8205, 0, 5386, 0, -3443, 0, 2120, 0, -1244, 0, 688, 0, -353, 0, 164, 0, -67, 0, 22, 0, -5];
Filter coefficient bit-width	19 bits
Sum of Filter coefficient values	524284
Passband Bandwidth	$0.4 * f_s$
Passband Ripple	+/- 0.001 dB (Max)
Stopband Rejection	95dB
Filter Type	Linear phase
Output Sample Rate	$f_s / 2$ Complex
Output Precision	16 Bits I, 16 Bits Q (input precision + 1-bit for filter overshoot + 1-bit for fraction)

Table 2: DHBF Filter Specification

Figure 3: DHBF Filter Response

DHBF OUTPUT FORMATS

The DHBF core outputs continuous complex data. The format of the data output is signed 2's-complement. Two different data outputs are available that allow direct interface with specific RFEL families of PFFT cores; the 'HiSpeed', 'QuadSpeed' and 'DualSpeed'.

HiSpeed Interface

The DHBF core has a multiplexed output that provides interleaved complex samples. This single output '*data_out*' operates at the core clock rate of f_s with alternate in-phase and quadrature-phase samples as shown in Figure 5.

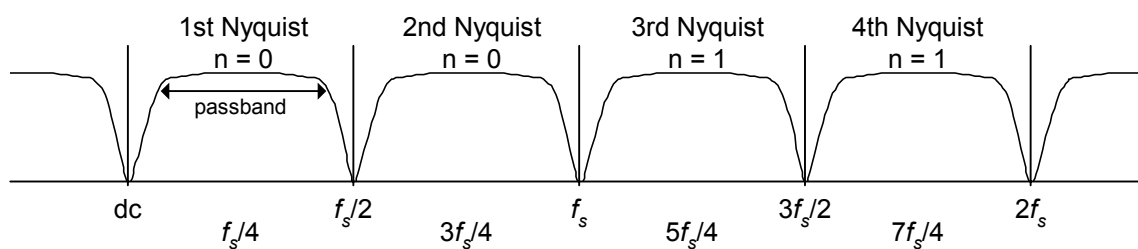
QuadSpeed or DualSpeed Interface

The DHBF core has a parallel output that provides complex samples at the complex sample rate of $f_s/2$, as shown in Figure 5

IF centre frequency / Nyquist Region

Figure 4 illustrates the centre frequency for each Nyquist region by way of formula. The '*ny_odd_even*' input to the core is used to select which Nyquist region is down-converted to base-band

- Odd Nyquist region, when '*ny_odd_even*' is logic '0' refers to the 1st Nyquist region or the symmetrical images of this (3rd, 5th etc).
- Even Nyquist region, when '*ny_odd_even*' is logic '1' refers to the 2nd Nyquist region or the symmetrical images of this (4th, 6th etc).



IF Input Frequency =

$(1+4n)f_s/4$ (1st Nyquist) or $(3+4n)f_s/4$ (2nd Nyquist).

Where n is a positive integer 0, 1, 2, etc.

Anti alias filtering prior to A to D is required to isolate only the nyquist frequencies of interest.

Figure 4: Nyquist Regions

It is important to note that the '*ny_odd_even*' signal is intended to be a 'static' input. If '*ny_odd_even*' is changed while the DHBF core is processing data, then output will be incorrect for the duration of the DHBF pipeline.

CORE INPUT SYNCHRONISATION

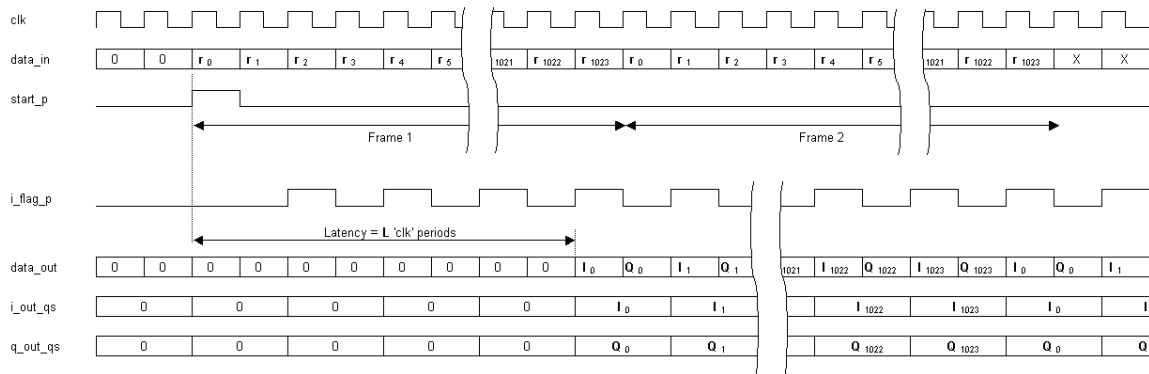


Figure 5: DHBF I/O Timing Diagram Example

CORE OUTPUT SYNCHRONISATION

Complex samples are output after the core latency period; the interleaved and parallel outputs are output with the same latency. The first sample is not specifically marked. The 'i_flag_p' signal provides an active high flag, coincident with each interleaved in-phase complex sample of 'data_out'. This 'i_flag_p' signal is also available for use as a clock enable for the parallel 'i_out_qs' and 'q_out_qs' outputs. This is shown in the timing diagram example of Figure 5.

PIPELINE LATENCY

Pipeline latency, defined as the time from when the first real sample is clocked into the DHBF to the time when the first processed complex output sample is clocked out from the DHBF, is shown in the timing diagram example of Figure 5.

The pipeline latency 'L' = 23 DHBF core clocks.

The DHBF core is a continuous pipeline, so after the initial delay due to the pipeline filling up as described previously, valid data is continuously available at the output.

CORE INTERFACE DESCRIPTION

All core inputs should be synchronised to the 'clk' signal. All core outputs are synchronised by the 'clk' signal.

Signal	Direction	Type	Width	Function
clk	IN	Std logic	1 bit	The core clock rate is equal to f_s . Where f_s is the real input rate.
rst_p	IN	Std logic	1 bit	An active-high pulse, of duration greater than 4 core clock periods, resets the DHBF control logic, but not the DHBF pipeline.
start_p	IN	Std logic	1 bit	Active-high pulse 1 core clock period in duration, marking the first sample of real input data.
ny_odd_even	IN	Std logic	1 bit	Signal that sets the DHBF operating region as odd Nyquist, when logic 0, else even Nyquist region. If changed, the pipeline must flush before valid data is output.
data_in	IN	Std logic vector	(13 downto 0)	2's complement real input data.
i_flag_p	OUT	Std logic	1 bit	Active-high signal asserted on alternate clock periods. Indicates each in-phase data sample out of the interleaved output. Or provides a clock enable for the parallel in-phase output, i_out_qs and quadrature-phase output, q_out_qs.
data_out	OUT	Std logic vector	(15 downto 0)	2's complement interleaved complex data samples.
i_out_qs	OUT	Std logic vector	(15 downto 0)	2's complement in-phase complex data samples.
q_out_qs	OUT	Std logic vector	(15 downto 0)	2's complement quadrature-phase complex data samples.

Table 3: DHBF Interface Specification

**Figure 6: DHBF Symbol****PLACE AND ROUTE REPORT SHOWING SILICON USAGE**

Release 5.2.03i - Map F.31

Xilinx Mapping Report File for Design 'dhbf_core'

Design Information

```
-----  
Command Line   : map -u -p xc2v3000-5fg676 dhbf_core.ngd -o dhbf_core_map.ncd  
dhbf_core.pcf  
Target Device  : x2v3000  
Target Package : fg676  
Target Speed   : -5  
Stepping Level : 1  
Mapper Version : virtex2 -- $Revision: 1.4 $  
Mapped Date    : Mon Dec 08 09:50:32 2003
```

Design Summary

```
-----  
Number of errors:      0  
Number of warnings:   83  
Logic Utilization:  
  Number of Slice Flip Flops:      2,789 out of 28,672    9%  
  Number of 4 input LUTs:          1,632 out of 28,672    5%  
Logic Distribution:  
  Number of occupied Slices:      1,541 out of 14,336   10%  
  Number of Slices containing only related logic: 1,541 out of 1,541 100%  
  Number of Slices containing unrelated logic:    0 out of 1,541    0%  
  *See NOTES below for an explanation of the effects of unrelated logic  
Total Number 4 input LUTs:        1,688 out of 28,672    5%  
  Number used as logic:            1,632  
  Number used as a route-thru:     10  
  Number used as Shift registers:   46
```

Total equivalent gate count for design: 44,917

Peak Memory Usage: 136 MB

NOTES:

Related logic is defined as being logic that shares connectivity -
e.g. two LUTs are "related" if they share common inputs.
When assembling slices, Map gives priority to combine logic that



is related. Doing so results in the best timing performance.

Unrelated logic shares no connectivity. Map will only begin packing unrelated logic into a slice once 99% of the slices are occupied through related logic packing.

Note that once logic distribution reaches the 99% level through related logic packing, this does not mean the device is completely utilized. Unrelated logic packing will then begin, continuing until all usable LUTs and FFs are occupied. Depending on your timing budget, increased levels of unrelated logic packing may adversely affect the overall timing performance of your design.

TIMING

Timing summary:

Timing errors: 0 Score: 0

Constraints cover 51007 paths, 0 nets, and 9402 connections (100.0% coverage)

Design statistics:

Minimum period: 9.355ns (Maximum frequency: 106.895MHz)

Analysis completed Mon Dec 08 09:51:59 2003

SIMULATED POWER DISSIPATION REPORT

Release 5.2.03i - XPower SoftwareVersion:F.31
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Design: dhbf_wrapper
Preferences: working\pandr\dhbf_wrapper.pcf
VCD File: working\vital_sim\dhbf_wrapper.vcd
Part: 2v3000fg676-5
Data version: ADVANCED 1.114 2002-12-13

Table with 3 columns: Power summary, I (mA), P (mW). Rows include Total estimated power consumption, Vccint 1.5V, Vccaux 3.3V, Vcco 3.3V, Clocks, Nets, Logic, Inputs, Outputs, and Quiescent 1.5V.



Quiescent 3.3V:	25	83
Quiescent 3.3V:	2	7

Note: The core power is indicated by the 'Logic' power only (535mW in this case). All other power contributions are due to the core being placed within a chip wrapper (Inputs, Outputs, Quiescents etc). The power simulation shown above was run using a clock period of 9.5 ns (105MHz) and 1 μ s of simulation; approximately 1052 random input samples.

VERIFICATION

Verification of the core is achieved in several distinct phases in the design cycle.

Initially, a bit-true Matlab model of the core is used to provide reference output data, using a specific set of input stimuli. These models exactly match the behaviour of the hardware architecture and their outputs are bit-true representations of the actual core.

The input stimuli are used to functionally (pre-synthesis) test the VHDL code using the ModelSim simulator. The results of these simulations are automatically compared with the reference data generated by the bit-true Matlab model and pass/fail indications are reported.

After the VHDL has been synthesised and place-and-route of the core is complete, timing simulations, based on the chip vendor's simulation netlist and gate delay files for the core are performed. These also take the form of the functional tests, but being based on the compiled netlist and timing files from the FPGA vendor, they verify the expected timings of the delivered core.

The bit-true Matlab models are also included.

TOOLS

The following tools and versions were used to generate this delivery.

Modelsim PE	Version: 5.4e
Leonardo Spectrum Level 2 or 3	Version: LS2003b_35
Xilinx ISE	Release Version: 5.2.03i Application Version: build+F-31+0
Matlab	Version: 6.5

DELIVERED FILE DIRECTORY STRUCTURE

The delivered file structure is shown in Figure 7.

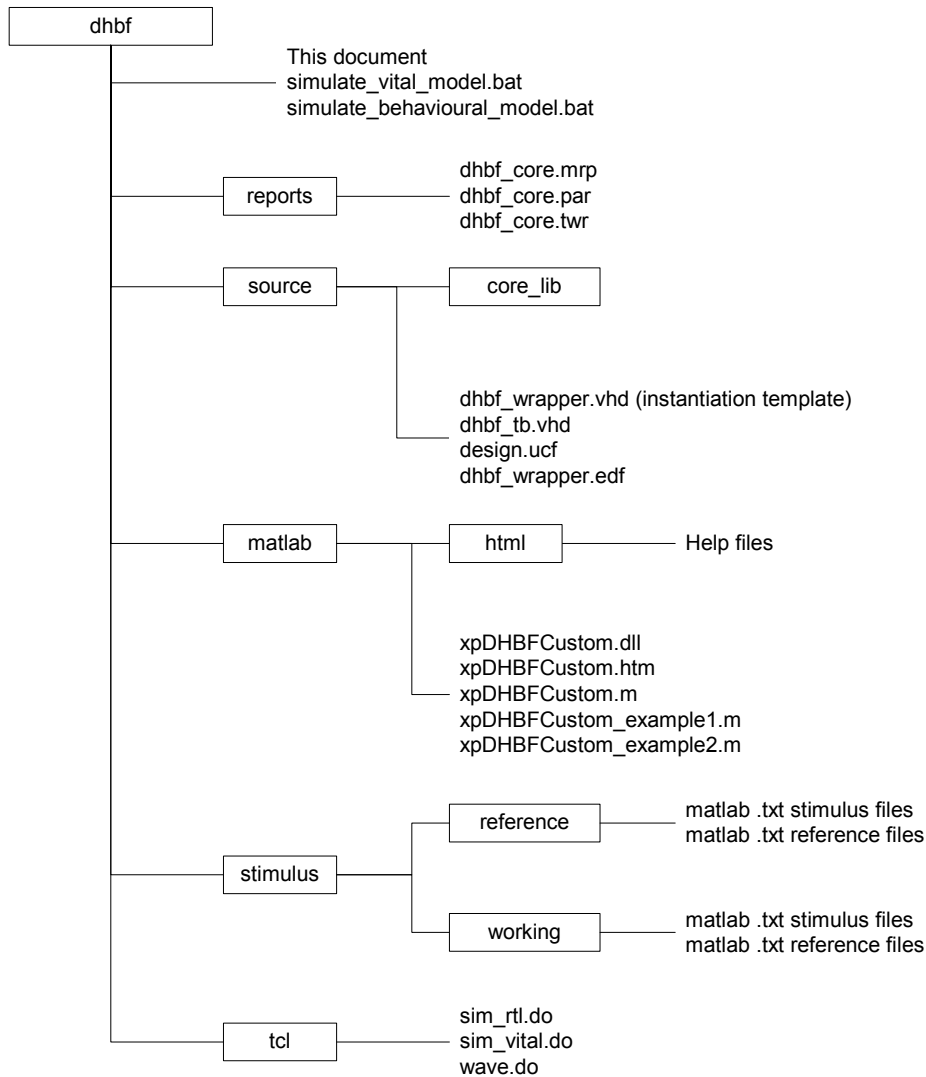


Figure 7: Delivered file structure

USING THE CORE

The following steps should be followed to ensure the EDIF netlist is included in the final design:

1. Include the instance(s) of the core within the target design (using '*dhbf/source/dhbf_wrapper.vhd*' as a reference).
2. Place the core EDIF file '*dhbf/source/dhbf_core.edf*' in the same directory as the final design EDIF, or add the macro search path in the process properties dialog box (right mouse click over implement design > properties) in XilinxISE.
3. Use the parameters contained in the constraints file '*dhbf/source/design.ucf*' as a reference.
4. Continue through normal place and route using the XilinxISE tools and the core will be automatically integrated into the design.

USING THE VHDL TEST BENCH

The VHDL test bench provides control and data stimuli to either the pre-compiled behavioural VHDL model, or a VITAL VHDL model that can be created by the XilinxISE tools. The Matlab bit-true model generated data stimuli and reference result files, held in the '*dhbf/stimulus/working*' directory. The test bench produces a latched pass / fail indication by automatically comparing the core outputs against the reference file values during simulation.

The name of the VHDL test bench is: - *dhbf_tb.vhd*

N.B. The dhbf_tb.vhd and dhbf_wrapper.vhd files are NOT to be edited by the user since all parameters are passed down via the simulation script.

RUNNING THE TEST BENCH FOR RTL SIMULATION

Running the DOS BAT file '*dhbf/simulate_behavioural_model.bat*' will perform a behavioural simulation. The script performs the following:

1. If the directory '*dhbf/working/behavioural_sim*' does not already exist (as will be the case the first time the script is run), then it is created.
2. The pre-compiled behavioural VHDL model '*dhbf/source/core_lib*' is copied into '*dhbf/working/behavioural_sim*'.
3. Modelsim is launched in GUI mode, and the TCL script '*dhbf/tcl/sim_rtl.do*' is run. Modelsim's transcript path is set to '*dhbf/working/behavioural_sim/transcript.txt*' so that it may be viewed after simulation if required.
4. When Modelsim has launched, the TCL script '*dhbf/tcl/sim_rtl.do*' performs the following:
 - a. Modelsim's working directory is set to '*dhbf/working/behavioural_sim*'.
 - b. A working library '*dhbf/working/behavioural_sim/lib*' is created and mapped to as '*work*'.
 - c. The copied version of the pre-compiled behavioural VHDL model library '*core_lib*' is mapped to as '*dhbf_core_library*', and updated (vcom -refresh) to the current ModelSim library format from it's delivered ModelSim 5.4e format.
 - d. The VHDL core wrapper '*dhbf/source/dhbf_wrapper.vhd*' and the VHDL test bench '*dhbf/source/dhbf_tb.vhd*' are compiled, and the test bench is loaded.
 - e. When the test bench is loaded, various generic values are assigned including the clock period of the core, input and output bit-widths and stimulus / reference result file paths (set to '*dhbf/stimulus/working*'). These paths and filenames determine if the test is for odd or even Nyquist verification.
 - f. The TCL script '*dhbf/source/wave.do*' is run to open a wave window with the relevant test bench signals required to check the simulation when completed.
 - g. The simulation is run until all of the test bench processes have reached their '*wait*' states.
5. The waveforms in the waveform window will illustrate the core signals and reference output results as multiplexed I and Q. The latched '*hs_sim_error*' and '*qs_sim_error*' signals should be a '0' to indicate no errors.
6. Several results files are also produced in the '*dhbf/stimulus/working*' directory, the samples are interleaved I and Q and separate files exist for both outputs.



7. Modify the TCL script *'dhbf/tcl/sim_rtl.do'* to select the other Nyquist region for verification.

RUNNING THE TEST BENCH FOR VITAL SIMULATION

Before VITAL simulation can take place, the user is required to edit and update the script *'dhbf/tcl/sim_vital.do'* to reflect the location of the compiled Xilinx *'simprim'* library. The following text within the script should be edited appropriately:

```
#####  
#### MAP TO YOUR COMPILED SIMPRIM LIBRARY HERE....  
vmap simprim E:/Xilinx/sim_lib/simprim  
#####
```

When this is done, VITAL simulation will be performed by running the DOS BAT file *'dhbf/simulate_vital_model.bat'*. This script performs the following:

1. If the directory *'dhbf/working/pandr'* does not already exist (as will be the case the first time the script is run), then the script performs the following:
 - a. The directory *'dhbf/working/pandr'* is created. This is where all of the wrapped core place and route files will be written.
 - b. *'dhbf/source/dhbf_wrapper.edi'* is implemented using the XilinxISE tools. This netlist is the core wrapper design, synthesised with I/O pads, I/O registers and a clock buffer.

The core *'dhbf/source/dhbf_core.edi'* is automatically included within the wrapper because *'dhbf/source'* has been specified as a macro search path within the script.
 - c. After implementation, the script checks to see which version of XilinxISE is currently installed. If version 5 is present, then the *'ngd2vhdl'* function is used to create the VHDL VITAL simulation model, otherwise it is assumed that the user has version 6 or later, and the newer *'netgen'* function is used instead.
2. If the directory *'dhbf/working/vital_sim'* does not already exist (as will be the case the first time the script is run), then it is created.
3. ModelSim is launched in GUI mode, and the TCL script *'dhbf/tcl/sim_vital.do'* is run. ModelSim's transcript path is set to *'dhbf/working/vital_sim/transcript.txt'* so that it may be viewed after simulation if required.
4. *'simulate_vital_model.bat'* now waits until the file *'dhbf/working/vital_sim/done'* is written by ModelSim to indicate that simulation is complete..
5. When ModelSim has launched, the TCL script *'dhbf/tcl/sim_vital.do'* performs the following:
 - a. Modelsim's working directory is set to *'dhbf/working/vital_sim'*.
 - b. A working library *'dhbf/working/vital_sim/lib'* is created and mapped to as *'work'*.
 - c. The compiled *'simprim'* library is mapped as specified by the user.
 - d. The *'dhbf/working/pandr/dhbf_wrapper_ba.vhd'* VITAL model and the VHDL test bench *'dhbf/source/dhbf_tb.vhd'* are compiled, and the test bench is loaded.
 - e. The SDF file *'dhbf/working/pandr/dhbf_wrapper_ba.sdf'* is applied to the unit under test *' uut'* when the test bench is loaded for simulation.
 - f. When the test bench is loaded, various generic values are assigned including the clock period of the core, input and output bit-widths and stimulus / reference result file paths, (set to *'dhbf/stimulus/working'*).

- g. The TCL script '*dhbf/source/wave.do*' is run to open a wave window with the relevant test bench signals required to check the simulation when completed.
 - h. The simulation is run for 1 μ s to flush the core with random data, then ModelSim is instructed to create a VCD file containing signal state information of all signals within the unit under test '*uut*'. The simulation is run for a further 10 μ s to allow >1000 samples of random data to pass through the core and be logged within the VCD file. When the simulation is complete, the file '*dhbf/working/vital_sim/done*' is created to tell '*dhbf/simulate_vital_model.bat*' to proceed with power analysis.
8. When '*dhbf /working/vital_sim/done*' has been written, the XilinxISE '*xpwr*' tool is launched to provide power analysis figures for the simulated VITAL design. The results of this power analysis are written to '*dhbf/working/vital_sim/dhbf_wrapper.pwr*', and are displayed on the Command Prompt window.
 9. Several results files are also produced in the '*dhbf/stimulus/working*' directory, the samples are interleaved I and Q and separate files exist for both outputs. A full simulation will not have been performed as this sim is targeted at power analysis only.
-

USING MATLAB MODELS

The core is provided with a Matlab model that matches the performance of the supplied core. Use of this model is described in the online help.

To display the online help for the model:

1. Open Matlab.
2. Navigate Matlab to the model installation directory: '*dhbf/matlab*'
3. Type '*xpDhbfCustom help*' at the command prompt.

The model is provided with some example scripts, which demonstrate its usage.

ADC	Analogue to Digital Converter
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
DFT	Discrete Fourier Transform
DHBF	Distributed Half-Band Filter
EDIF	Electronic Data Interchange Format
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
I/O	Input / Output
LSB	Least Significant Bit
MSB	Most Significant Bit
MS/s	Million Samples Per Second
PFT	Pipelined Frequency Transform
PFFT	Pipelined Fast Fourier Transform
RFEL	RF Engines Limited
RTL	Register Transfer Level
RPM	Relationally Placed Macro
UCF	User Constraints File
VHDL	Very High Speed IC Hardware Description Language
VITAL	VHDL Initiative Toward ASIC Libraries
.bat	Batch File
.do	Modelsim script file
.vhd	A VHDL File
.edf	An EDIF File
.ucf	User Constraints File
.mrp	Map Report File
.par	Place and Route Report File
.sdf	Standard Delay Format
.twr	Place and Route Timing Report File

Table 4: Glossary