UniBoard Wideband-FFT Module Description

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Harm Jan Pepping	ASTRON	1 October 2012
Controle / Checked: Eric Kooistra	ASTRON	
Goedkeuring / Approval: Andre Gunst	ASTRON	
Autorisatie / Authorisation: **Handtekening / Signature** Andre Gunst	ASTRON	

Distribution list:

Group:	Others:
Andre Gunst Eric Kooistra Daniel van der Schuur	Gijs Schoonderbeek Sjouke Zwier Harro Verkouter (JIVE) Jonathan Hargreaves (JIVE) Salvatore Pirruccio (JIVE)

Document history:

Revision	Date	Author	Modification / Change
0.1	2012-9-5	Harm Jan Pepping	Creation
0.2	2012-10-1	Harm Jan Pepping	Added content for first review
0.3	2013-4-3	Harm Jan Pepping	Processed comments of Eric Kooistra. Added quantization section 5.4. Added python verification section 6.3

Table of contents:

Terminology:

DIAG	Diagnostics (VHDL module)
DP	Data Path (VHDL module)
DSP	Digital Signal Processing
DUT	Device Under Test
EOP	End Of Packet
FFT	Fast Fourier Transformation
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IO	Input Output
MISO	Master In Slave Out
MM	Memory-Mapped
MOSI	Master Out Slave In
Nof	Number of
RAM	Random Access Memory
Signal Path	Time series signal
SISO	Source In Sink Out
SOP	Start Of Packet
SOPC	System On a Programmable Chip (Altera)
SOSI	Source Out Sink In
SRC	Source
ST	Streaming
Subband	Frequency signal

Definitions:

N	FFT number of points
P	Wideband factor = sample clock frequency/DSP clock frequency
M	N/P = number of FFT points per wideband section

References:

1. ‘APERTIF Filter Bank Firmware Specification Part 2’, ASTRON-SP-054, Eric Kooistra
2. ‘A Radix-2 Single Delay Feedback (R2SDF) architecture based generic Fast Fourier Transform for firmware implementation’, ASTRON-RP-755, R.T. Rajan
3. $UNB/Firmware/dsp/rTwoSDF/
4. $UNB/Firmware/dsp/fft/
5. ‘Understanding Digital Signal Processing’, R. Lyons
6. ‘BN Capture Design Description’, ASTRON-RP-498, Eric Kooistra, Daniel van der Schuur
7. ‘DP Streaming Module Description’, ASTRON-RP-382, Eric Kooistra
8. $UNB/Firmware/modules/Lofat/st/
9. $UNB/Firmware/dsp/fft/tb/vhdl/
10. $UNB/Firmware/dsp/rTwoSDF/tb/data/
11. ‘DIAG Module Description’, ASTRON-RP-1313, Eric Kooistra, Harm Jan Pepping, Daniel van der Schuur
12. $UPE/apps/bn_fb/
13. ‘ADU Handler Module Description’, ASTRON-RP-1323, Eric Kooistra

1 Introduction

1.1 Purpose

The fft_wide unit performs a N-point Wideband FFT(Fast Fourier Transformation) on data that is partly applied in serial and partly applied in parallel. The fft_wide unit specifically suits applications where the sample clock is higher than the DSP processing clock. For each output stream a subband statistic unit is included which can be read via the memory mapped interface.

1.2 Module overview

An overview of the fft_wide unit is shown in Figure 1. The fft_wide unit calculates a N-point FFT and has P number of input streams. Data of each input is offered to a M-point pipelined FFT, where M=N/P. The output of all pipelined FFTs is then connected to a P-point parallel FFT that performs the final stage of the wideband FFT. Each output of the parallel FFT is connected to a subband statistics unit that calculates the power in each subband. The MM interface is used to read out the subband statistics.

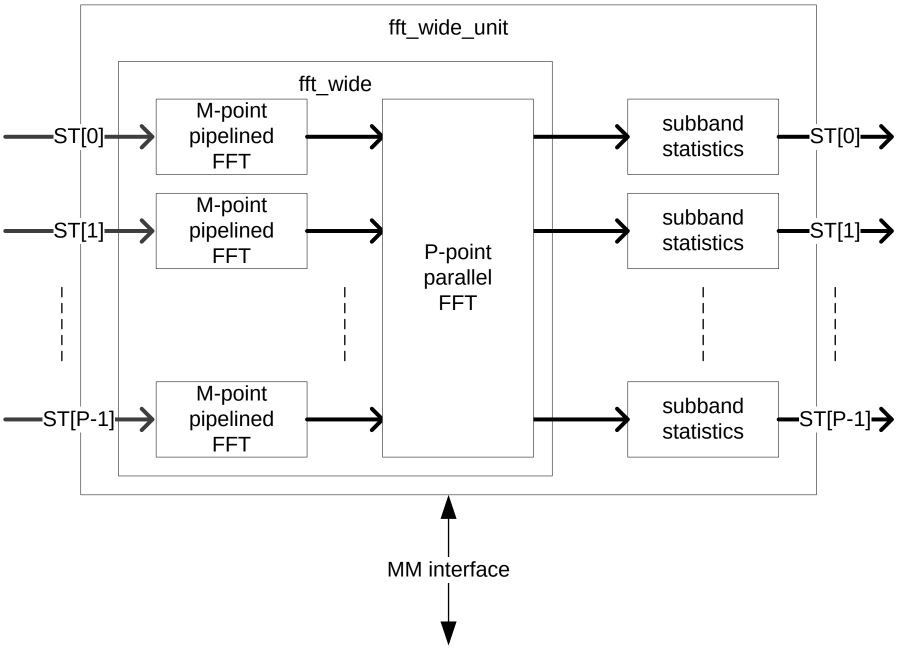


Figure 1: FFT Wide unit overview

1.3 rTwoSDF_lib and fft_lib

At the start of the development of the wideband FFT a pipelined FFT was already developed as described in [1] and [2]. All design files that relate to the development of the rTwoSDF design reside in the rTwoSDF_lib as pointed by [3]. The rTwoSDF pipelined FFT design is used as building block for the development of the wideband extension. All design files regarding the wideband extension are located in the fft_lib as pointed by [4].

2 Firmware interface

This chapter covers all firmware interface related topics of the fft_wide unit. It describes the functionality of the in- and output ports.

2.1 Clock domains

There are two clock domains used in the fft_wide unit: the mm_clk and the dp_clk domain. Figure 2 shows an overview of the clock domains in the fft_wide unit. The only unit that is connected to both clock domains is the memory of the subband statistics module. This memory is a dual ported ram that holds the results of the subband statistics. Table 1 lists both clocks and their characteristics.

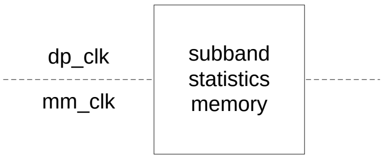


Figure 2: fft_wide_unit clock domains

Name	Frequency (MHz)	Description
DP_CLK	200 MHz	Clock for the datapath
MM_CLK	125 MHz	Clock for the memory mapped interface.

Table 1: fft_wide unit clock signals

2.2 Parameters

The parameters that define an instantiation of the fft_wide unit are grouped in three VHDL records as listed in Table 2.

Generic	Type	Description
g_fft	t_fft	This record in defined in fft_pkg.vhd. It contains parameters that define the behavioural of the FFT. Content of the t_fft record is detailed in Table 3.
g_pft_pipeline	t_fft_pipeline	This record contains pipeline settings for the parallel FFTs. The record is defined in rTwoSDFPkg.vhd, which is located in the rTwoSDF library. More info can be found in 2.3.3 of [1] and chapter 3 in [2]. Content of the t_fft_pipeline record is listed in Table 4.
p_fft_pipeline	t_fft_pipeline	This record contains pipeline settings for the pipelined FFTs. The record is defined in rTwoSDFPkg.vhd, which is located in the rTwoSDF library. More info can be found in 2.3.3 of [1] and chapter 3 in [2]. Content of the t_fft_pipeline record is listed in Table 4.

Table 2 fft_wide unit parameters

The items of the t_fft record are listed in Table 3. The column named “value” specifies the default value that is used.

Generic	Type	Value	Description
use_reorder	BOOLEAN	true	When set to ‘true’, the output bins of the FFT are reordered in such a way that the first bin represents the lowest frequency and the highest bin represents the highest frequency.
use_separate	BOOLEAN	true	When set to ‘true’ a separate algorithm will be enabled in order to retrieve two separate spectra from the output of the complex FFT in case both the real and imaginary input of the complex FFT are fed with two independent real signals.
nof_chan	NATURAL	0	Defines the number of channels (=time-multiplexed input signals). The number of channels is 2^{nof_chan}. Multiple channels is only supported by the pipelined FFT.
wb_factor=P	NATURAL	4	The number that defines the wideband factor. It defines the number of parallel pipelined FFTs.
twiddle_offset	NATURAL	0	The twiddle offset is used for the pipelined sections in the wideband configuration. This is explained in detail in 5.3.
nof_points=N	NATURAL	1024	The number of points of the FFT.
in_dat_w	NATURAL	8	Width in bits of the input data. This value specifies the width of both the real and the imaginary part.
out_dat_w	NATURAL	14	The bitwidth of the real and imaginary part of the output of the FFT. The relation with the in_dat_w is as follows: out_dat_w = in_dat_w + (log2(nof_N))/2+1
stage_dat_w	NATURAL	18	The bitwidth of the data that is used between the stages (= DSP multiplier-width)
guard_w	NATURAL	2	Number of bits that function as guard bits. The guard bits are required to avoid overflow in the first two stages of the FFT.
guard_enable	BOOLEAN	true	When set to ‘true’ the input is guarded during the input resize function, when set to ‘false’ the input is not guarded, but the scaling is not skipped on the last stages of the FFT (based on the value of guard_w).
stat_data_w	POSITIVE	56	Width of the output of the subband statistics unit. This value must be high enough to accommodate the highest possible bitgrowth in the subband statistic unit. Highest integration period is set to one second.
stat_data_sz	POSITIVE	2	This number specifies how many 32-bit registers are required to read out one accumulated value.

Table 3: t_fft record fields

The items of the t_fft_pipeline record are listed in Table 4. The column named “value” specifies the default value.

Generic	Type	Value	Description
stage_lat	NATURAL	1	The output latency of a pipelined fft stage or a parallel fft butterfly.
weight_lat	NATURAL	1	Latency that is required to look up the addressed weight factors.
mul_lat	NATURAL	3	The latency of the multipliers that are used in the butterflies.
bf_lat	NATURAL	1	Output latency of a pipelined or an internal latency in the parallel butterfly.
bf_use_zdly	NATURAL	1	Enable for the usage of the feedback delay lines bf_in_a_zdly and bf_out_d_zdly.
bf_in_a_zdly	NATURAL	0	Size for extra delay to be added to the feedback delay line that drives input a of the rTwoBF units. (Only applicable to pipelined stages)
bf_out_d_zdly	NATURAL	0	Size for extra delay to be added to the output d of othe rTwoBF units. (Only applicable to pipelined stages)

sep_lat	NATURAL	1	Latency that is introduced by the separate function.

Table 4 t_fft_pipeline record fields

2.3 Interface signals

The interface signals of the fft_wide unit are shown in Figure 3 and Table 5 lists the general specifications of these interfaces. More detailed information about the interfaces can be found in the following paragraphs.

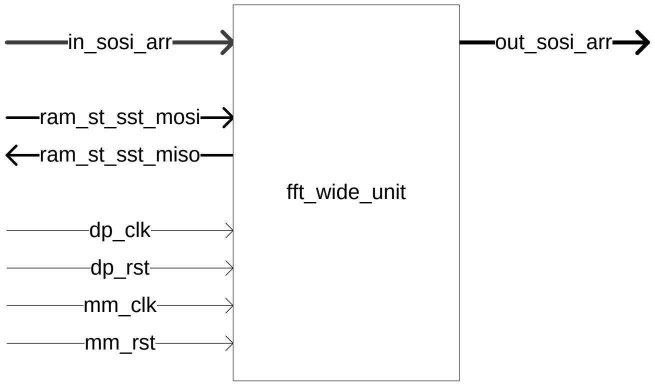


Figure 3: interface signals

Interface	Type	Size or Span	Description
in_sosi_arr	t_dp_sosi_arr	wb_factor	Array of input streams where each stream holds 1/wb_factor portion of the time-domain input data. Data can be single complex or dual real. Format of the data is explained in detail in 2.3.1.
out_sosi_arr	t_dp_sosi_arr	wb_factor	Array of output streams containing the frequency-domain data. Format of the data is explained in detail in 2.3.2.
ram_st_sst_mosi	t_mem_mosi	nof_points	A mosi interface to read out the subband statistics data.
ram_st_sst_miso	t_mem_miso	nof_points	A miso interface to read out the subband statistics data.
dp_clk	std_logic	na	Datapath clock
dp_rst	std_logic	na	Datapath reset
mm_clk	std_logic	na	Memory mapped interface clock
mm_rst	std_logic	na	Memory mapped interface reset

Table 5: interface signals

2.3.1 IN_SOSI_ARR interface

The in_sosi_arr is an array of streams where each stream holds 1/P part of the time domain data of one complex input (x) or two real inputs (a and b). The RE and IM field of the sosi record are used. For a fft_wide unit with a wb_factor of P, the time-domain data is divided over the streams as follows:

- Input index t is p + [0, P, 2P, …, (M-1)P]

  in_sosi_arr[0].re = x_{re}(t) or a(t) for p = 0
  in_sosi_arr[1].re = x_{re}(t) or a(t) for p = 1
  …
  in_sosi_arr[P-1].re = x_{im}(t) or a(t) for p = P-1

in_sosi_arr[0].im = $x_{im}(t)$ or b(t) for p = 0
in_sosi_arr[1].im = $x_{im}(t)$ or b(t) for p = 1
…
in_sosi_arr[P-1].im = $x_{im}(t)$ or b(t) for p = P-1

The input streams consist of blocks that are marked by the sop and the eop fields of the sosi record. The number of samples in a packet (M) is determined by N and P (M = N/P). In case two real inputs are used, the first real input (a) should be connected to the RE field of the sosi record and the second real input (b) should be connected to the IM field of the sosi record. Figure 4 shows the packet format of a single stream for the complex option (x) and the two real inputs option (a and b). Note that gaps within a packet and gaps in between packets are allowed (valid signal goes low).

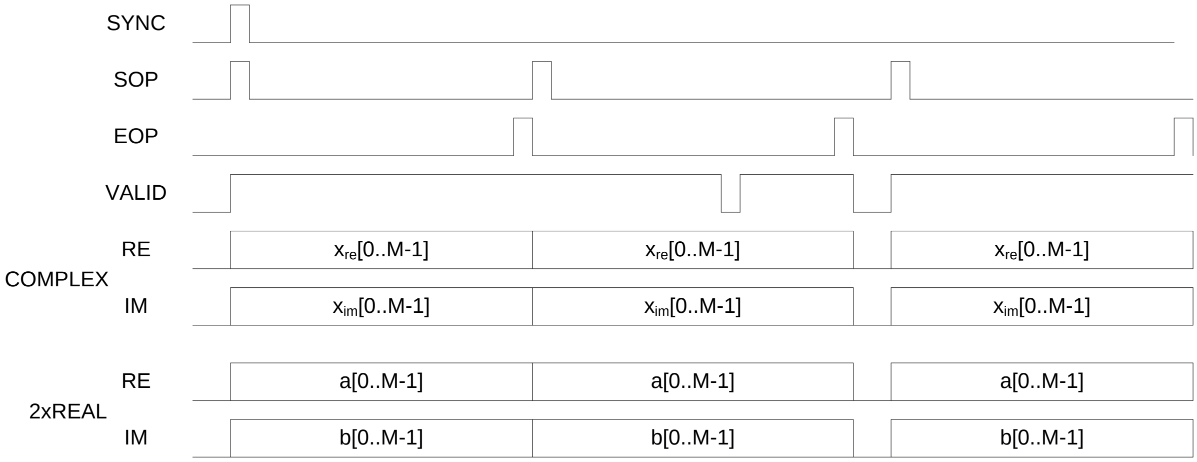


Figure 4 Input packet format fft_wide unit

Note that the fft_wide unit does not provide a corresponding in_siso_arr, because there is no need for back pressure. The fft_wide unit can always receive data.

2.3.2 OUT_SOSI_ARR interface

The output of the fft_wide unit is also composed of P streams. Each stream carries 1/P part of the spectrum. Provided that the reorder option is enabled, the frequency domain data is divided over the streams as follows:

- For a complex input signal, index f is p + [0, P, 2P, …, (M-1)P]

  out_sosi_arr[0].re = $X_{re}(f)$ for p = 0
  out_sosi_arr[1].re = $X_{re}(f)$ for p = 1
  …
  out_sosi_arr[P-1].re = $X_{re}(f)$ for p = P-1

  The same counts for the imaginary part:

  out_sosi_arr[0].im = $X_{im}(f)$ for p = 0
  out_sosi_arr[1].im = $X_{im}(f)$ for p = 1
  …
  out_sosi_arr[P-1].im = $X_{im}(f)$ for p = P-1

- For 2x real input signals, index f is [p,p] + [0, 0, P, P, 2P, 2P, …, (M-1)P/2, (M-1)P/2]

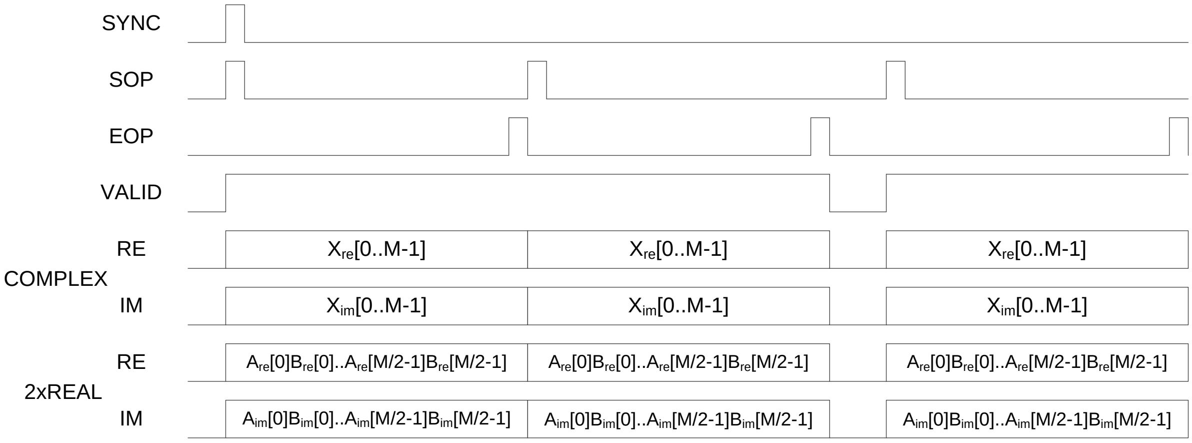
  out_sosi_arr[0].re = $A_{re}(f)$, $B_{re}(f)$ for [p,p] = [0,0]
  out_sosi_arr[1].re = $A_{re}(f)$, $B_{re}(f)$ for [p,p] = [1,1]

…
out_sosi_arr[P-1].re = $A_{re}(f)$, $B_{re}(f)$ for [p,p] = [P-1,P-1]

The same counts for the imaginary part:

out_sosi_arr[0].im = $A_{im}(f)$, $B_{im}(f)$ for [p,p] = [0,0]
out_sosi_arr[1].im = $A_{im}(f)$, $B_{im}(f)$ for [p,p] = [1,1]
…
out_sosi_arr[P-1].im = $A_{im}(f)$, $B_{im}(f)$ for [p,p] = [P-1,P-1]

The content of the output of the fft_wide unit blocks differs per configuration. For both the complex and 2x real configuration the block content of a single stream is shown in Figure 5. In both cases the reorder function is enabled and in the 2x real configuration the separate function is enabled as well.



SYNC
SOP
EOP
VALID
RE
COMPLEX
IM
RE
2xREAL
IM
X_{re}[0..M-1]
X_{im}[0..M-1]
$A_{re}[0]B_{re}[0]..A_{re}[M/2-1]B_{re}[M/2-1]$
$A_{im}[0]B_{im}[0]..A_{im}[M/2-1]B_{im}[M/2-1]$

Figure 5 Output packet format fft_wide unit

Note that there can be gaps in between two consecutive blocks, but a block is always uninterrupted due to the reorder buffering. The output arrays are not accompanied with a corresponding siso array, since the fft_wide unit is not required to cope with backpressure.

2.3.3 RAM_ST_SST_MOSI interface

The subband statistics can be read via the ram_st_sst_mosi interface. The address span of this interface is determined by stat_data_sz*N. Table 6 shows the interface signals where N = 1024 and stat_data_sz = 2.

Signal	Type	Description
ram_st_sst_mosi.address[10:0]	MOSI	Word address range for mm_bst_mosi interface, supporting 2*1024 = 2048 registers.
ram_st_sst_mosi.wrdata[31:0]	MOSI	Write data word, must be valid when wr is asserted.
ram_st_sst_mosi.wr	MOSI	Write strobe.
ram_st_sst_mosi.rd	MOSI	Read strobe.
ram_st_sst.rddata[31:0]	MISO	Read data word which is valid one clock cycle after assertion of rd.

Table 6 ram_st_sst_mosi interface

2.3.4 Clocks and resets

Table 7 shows an overview of the clocks and reset signals that are available on the fft_wide unit.

Signal	Type	Description
dp_clk	Clock	Clock input for the datapath interface of the fft_wide unit.
dp_rst	Reset	Reset input for the datapath clock domain registers.
mm_clk	Clock	Clock input for the memory mapped interface parts of the fft_wide unit.
mm_rst	Reset	Reset input for the memory mapped interface parts of the fft_wide unit.

Table 7 Clocks and resets

3 Software interface

This chapter describes the software interface for the fft_wide unit. The fft_wide unit contains one register span that contains the registers that contain the subband statistics data. The size of the subband statistics span depends on the value of the parameters that are used to configure the fft_wide unit.

3.1 Subband statistics span

For every stream in the OUT_SOSI_ARR the fft_wide unit contains a subband statistics unit. The memory interfaces of all these P subband statistics units are merged into one. Therefor the subband statistics can be considered as one single unit. The subband statistics unit estimates the power of each subband value and integrates these values during a sync period. When a sync period has expired (in other words: a sync pulse is applied) the registers will be updated with the new integrated power values. The number of registers is determined by stat_data_sz*N. The order of the registers is determined by P. In the Apertif system stat_data_sz = 2, N = 1024 and P = 4. This leads to 2048 registers which are grouped in 4x512 register blocks which are partly listed in Table 8. Note that each subband statistic value is spread over two 32-bit registers. The actual bit width of a subband statistic is set via parameter stat_data_w, which is set to 56 in the Apertif system. This means that the upper 8 bits of the “up” register can be discarded.

Name	Address (words)	Size (words)	Read/ Write	Description
str_0_sbst_0_low	0x0	1	r/w	32 lsb’s of power in stream 0 on subband 0
str_0_sbst_0_up	0x1	1	r/w	32 msb’s of power in stream 0 on subband 0
str_0_sbst_1_low	0x2	1	r/w	32 lsb’s of power in stream 0 on subband 1
str_0_sbst_1_up	0x3	1	r/w	32 msb’s of power in stream 0 on subband 1
str_0_sbst_2_low	0x4	1	r/w	32 lsb’s of power in stream 0 on subband 2
str_0_sbst_2_up	0x5	1	r/w	32 msb’s of power in stream 0 on subband 2
-----------------------	------	---	-----	--
str_0_sbst_255_low	0x1FE	1	r/w	32 lsb’s of power in stream 0 on subband 255
str_0_sbst_255_up	0x1FF	1	r/w	32 msb’s of power in stream 0 on subband 255
str_1_sbst_0_low	0x200	1	r/w	32 lsb’s of power in stream 1 on subband 0
str_1_sbst_0_up	0x201	1	r/w	32 msb’s of power in stream 1 on subband 0
str_1_sbst_1_low	0x202	1	r/w	32 lsb’s of power in stream 1 on subband 1
str_1_sbst_1_up	0x203	1	r/w	32 msb’s of power in stream 1 on subband 1
-----------------------	------	---	-----	--
str_1_sbst_255_low	0x3FE	1	r/w	32 lsb’s of power in stream 1 on subband 255
str_1_sbst_255_up	0x3FF	1	r/w	32 msb’s of power in stream 1 on subband 255
str_2_sbst_0_low	0x400	1	r/w	32 lsb’s of power in stream 2 on subband 0
str_2_sbst_0_up	0x401	1	r/w	32 msb’s of power in stream 2 on subband 0
str_2_sbst_1_low	0x402	1	r/w	32 lsb’s of power in stream 2 on subband 1
str_2_sbst_1_up	0x403	1	r/w	32 msb’s of power in stream 2 on subband 1
-----------------------	------	---	-----	--
str_2_sbst_255_low	0x5FE	1	r/w	32 lsb’s of power in stream 2 on subband 255
str_2_sbst_255_up	0x5FF	1	r/w	32 msb’s of power in stream 2 on subband 255
str_3_sbst_0_low	0x600	1	r/w	32 lsb’s of power in stream 3 on subband 0
str_3_sbst_0_up	0x601	1	r/w	32 msb’s of power in stream 3 on subband 0
str_3_sbst_1_low	0x602	1	r/w	32 lsb’s of power in stream 3 on subband 1
str_3_sbst_1_up	0x603	1	r/w	32 msb’s of power in stream 3 on subband 1
-----------------------	------	---	-----	--
str_3_sbst_255_low	0x7FE	1	r/w	32 lsb’s of power in stream 3 on subband 255
str_3_sbst_255_up	0x7FF	1	r/w	32 msb’s of power in stream 3 on subband 255

Table 8 subband statistics span

4 Module Design

4.1 Algorithm

4.1.1 FFT

There are many books about the FFT algorithm. A good introduction can be found in chapter 4 of [5].

4.1.2 Separate

The separate algorithm is well explained in chapter 13.5.1 of [5].

4.2 Architecture

Several subdesigns have been defined in order to create the eventual wideband FFT. The architectures of the necessary units are provided in this chapter. The wideband FFT specifications have led to the following (sub)-designs:

- Complex Pipelined FFT for two real inputs: fft_r2_pipe
- Complex Parallel FFT for two real inputs: fft_r2_par
- Complex Wideband FFT for two real inputs: fft_r2_wide

The '_r2' denotes radix-2 (see [2]).

4.2.1 fft_r2_pipe

The architecture for a pipelined FFT is based on design units from the rTwoSDF_lib and is basically the same as the rTwoSDF unit. Detailed information of the rTwoSDF unit and its pipelined aspects can be found in [1]. The difference with respect to the rTwoSDF unit is that the fft_r2_pipe unit must be capable of processing two real inputs as well. Therefor the architectural block diagram is extended with an optional separate function. An architectural overview of the fft_r2_pipe is given in Figure 6.

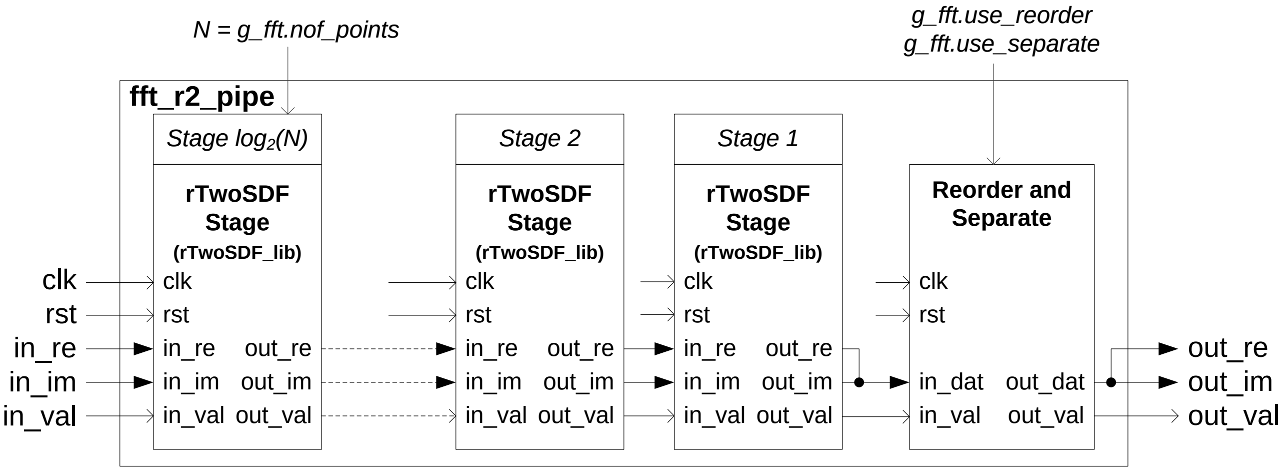


Figure 6 Architecture of the fft_r2_pipe unit

4.2.2 fft_r2_par

In case of a parallel FFT all time domain samples for a slice come in parallel and therefor all multiplications and additions have to be performed in parallel as well. The architecture for a parallel FFT is shown in Figure 7. In Figure 7 the number of points is set to 16. Each square represents an optimized complex butterfly. The numbers in the butterflies refer to the exponent *k* in W_N^k, the twiddle factors. The inputs and outputs of the butterfly are shown in Figure 8. The parallel FFT is also capable of reordering the output data and processing two real inputs. Therefore a parallel reorder and parallel separate function are defined as well.

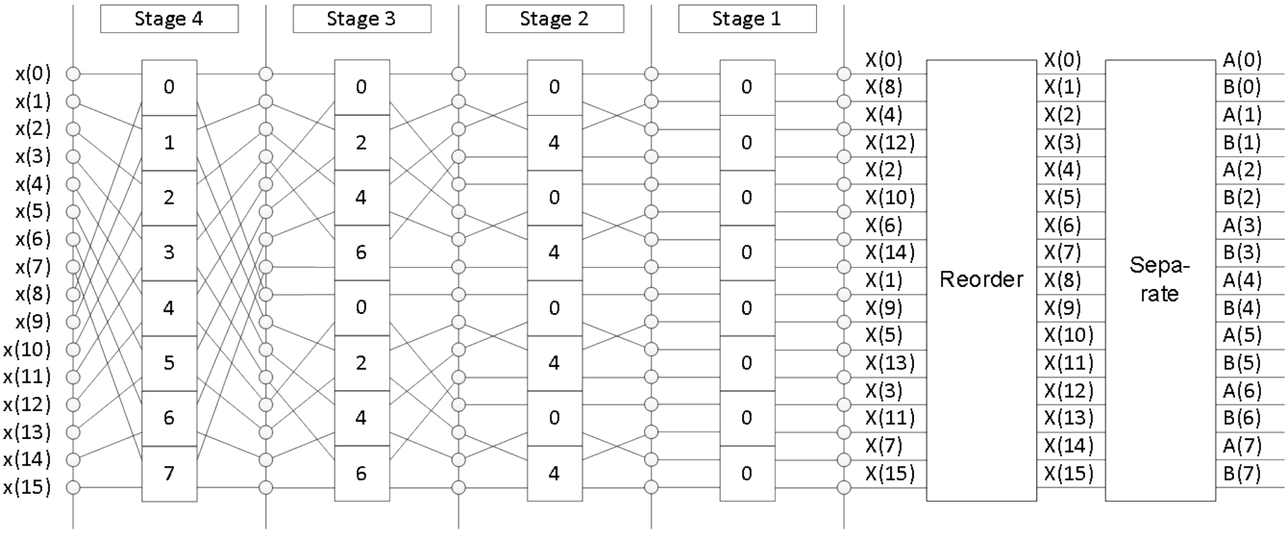


Figure 7 Architecture of the fft_r2_par unit (shown for nof_points = 16)

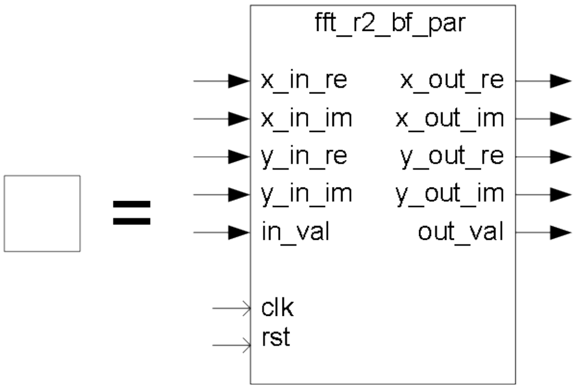


Figure 8 Inputs and outputs of the parallel butterfly

4.2.3 fft_r2_wide

The wideband variant of the FFT is partly pipelined and partly composed in parallel. The amount of parallelization is specified by the P (wideband factor). For the Apertif beamformer P is set to 4. A schematic overview of the architecture of the wideband FFT is shown in Figure 9. The reorder functionality is inherited from both the fft_r2_par and fft_r2_pipe units, but for the separation functionality a dedicated wideband variant must be designed. More detailed information about the architecture of the wideband FFT can be found in chapter 2.5 of [1].

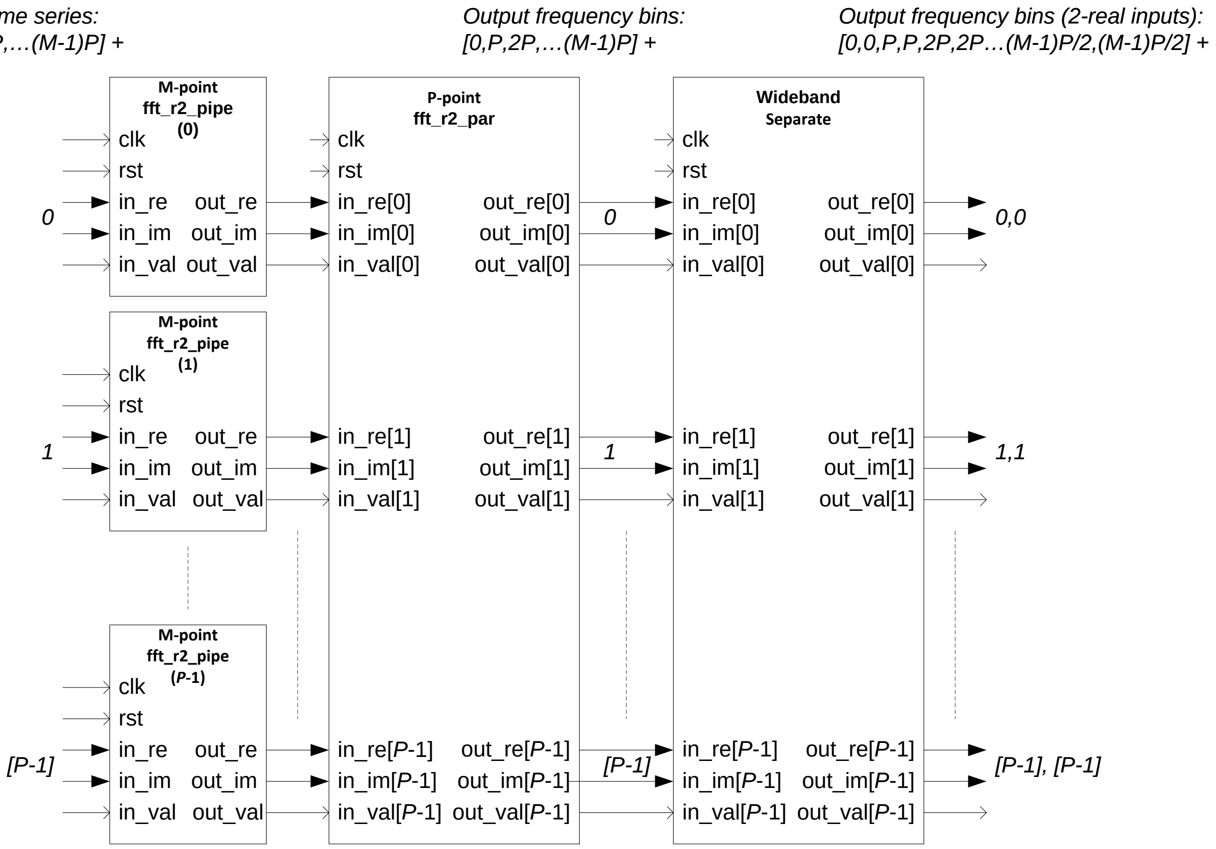


Figure 9 Architecture of the wideband FFT

5 Implementation

This chapter describes the implementation of the fft_wide unit. During the development of the fft_wide unit design several building blocks were created as defined in [1]. The implementation of all these intermediate blocks and the eventually fft_wide design are described in detail in the following paragraphs.

5.1 fft_r2_pipe

The fft_r2_pipe unit consists of two parts: the rTwoSDF stages and the Reorder and Separate functionality as shown in Figure 6.

5.1.1 rTwoSDF stages

The rTwoSDF stages are instantiated from the rTwoSDF_lib using a generate statement. The number of stages is determined by log2(nof_points). Detailed information about the usage of the design units from the rTwoSDF_lib can be found in [1] and [2].

5.1.2 fft_reorder_sepa_pipe

To facilitate the optional reordering and separation a dedicated unit is designed that can do both. The fft_reorder_sepa_pipe unit is based on a dual paged ram in order to support both functions. A schematic representation of the fft_reorder_sepa_pipe unit can be found in Figure 10. The write-side consists of a counter that drives the write address port of the ram. The in_val signal is used to drive the write enable port. A page turn is forces when the counter has reached the value nof_points-1 and the in_val signal is active. The read-side is driven by a read process that drives the read enable and the read address. A dedicated separation unit (fft_sepa) is used that performs the actual separation algorithm.

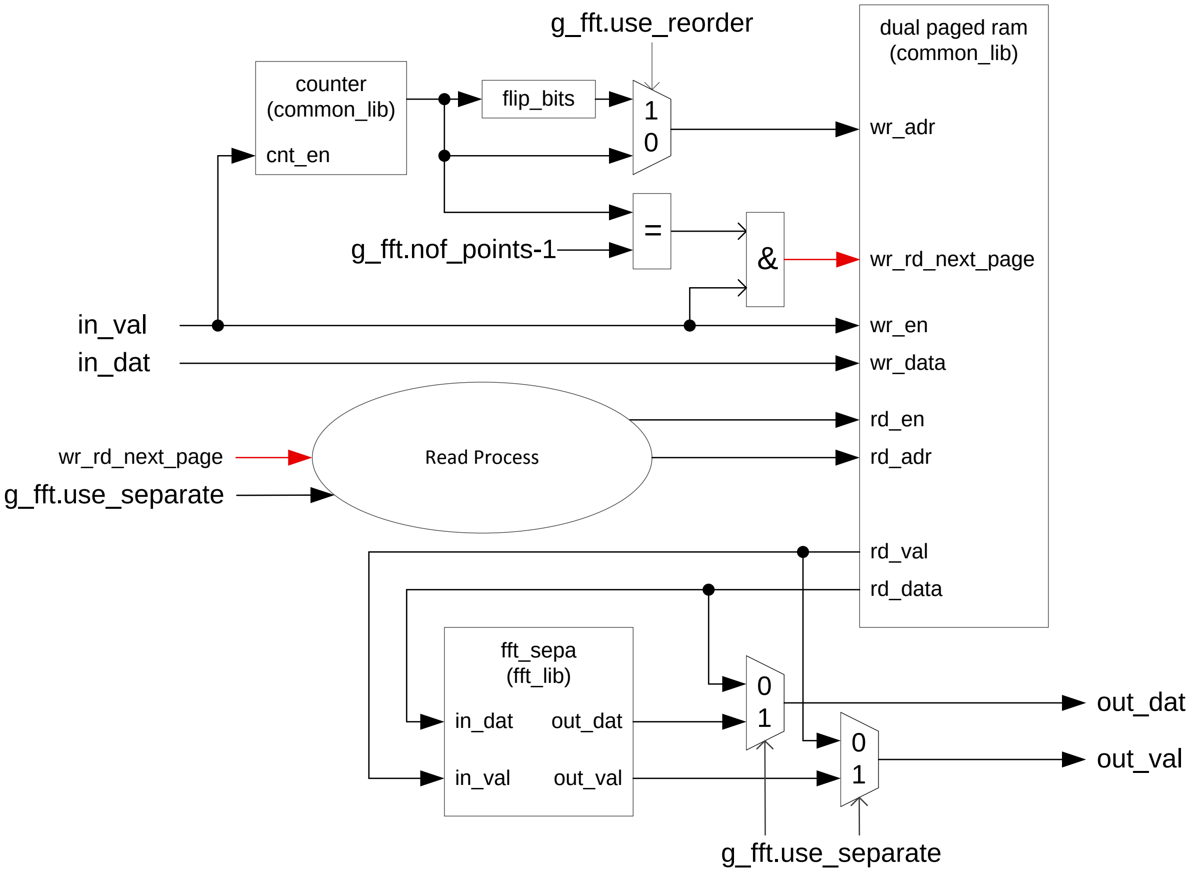


Figure 10 Block diagram of the fft_reorder_sepa_pipe unit

5.1.2.1 Reordering

The reorder functionality is accomplished by bit-flipping the write address for normal output order. The address is not flipped when a reversed output order is required. More information about the bit reversal can be found in chapter 4.4 of [5].

5.1.2.2 Read process

The read process is triggered by the wr_rd_next_page signal and generates the read address for the RAM. When separation is disabled an incrementing counter is used to drive the read address. When separation is enabled a different type of addressing is used in order to supply the fft_sepa unit with the correct data. Based on the separation algorithm the fft_sepa unit expects the data in an interleaved order. For a 1024-point FFT this is:

X(0), X(1024), X(1), X(1023), X(2) ….. X(511), X(512).

Since modulo(N) addressing is used the value of X(1024)=X(0). This sequence is established in the read process by using an incrementing counter and a decrementing counter that are fed to the read address port in an interleaved way.

5.1.2.3 fft_sepa

The fft_sepa unit performs the actual separation algorithm in order to create the interleaved output that corresponds to two real input signals. Figure 11 shows a schematic overview of the fft_sepa unit that consists of an adder and a subtractor that are used to create the following terms, where N = nof_points and m = frequency bin:

$$
\begin{aligned}
2A_{re}(m) &= X_{re}(N\text{-}m) + X_{re}(m)\\
2A_{im}(m) &= X_{im}(m) - X_{im}(N\text{-}m)\\
2B_{re}(m) &= X_{im}(m) + X_{im}(N\text{-}m)\\
2B_{im}(m) &= X_{re}(N\text{-}m) - X_{re}(m)
\end{aligned}
$$

All multiplexers and de-multiplexers are controlled by the same switch signal (S) that toggles on every valid data input. To retrieve the values of A_{re}, A_{im}, B_{re} and B_{im} the results of the adder and subtractor (sub-res-reg and add-res-reg) are shifted one step to the right before they are placed in the output register (output reg).

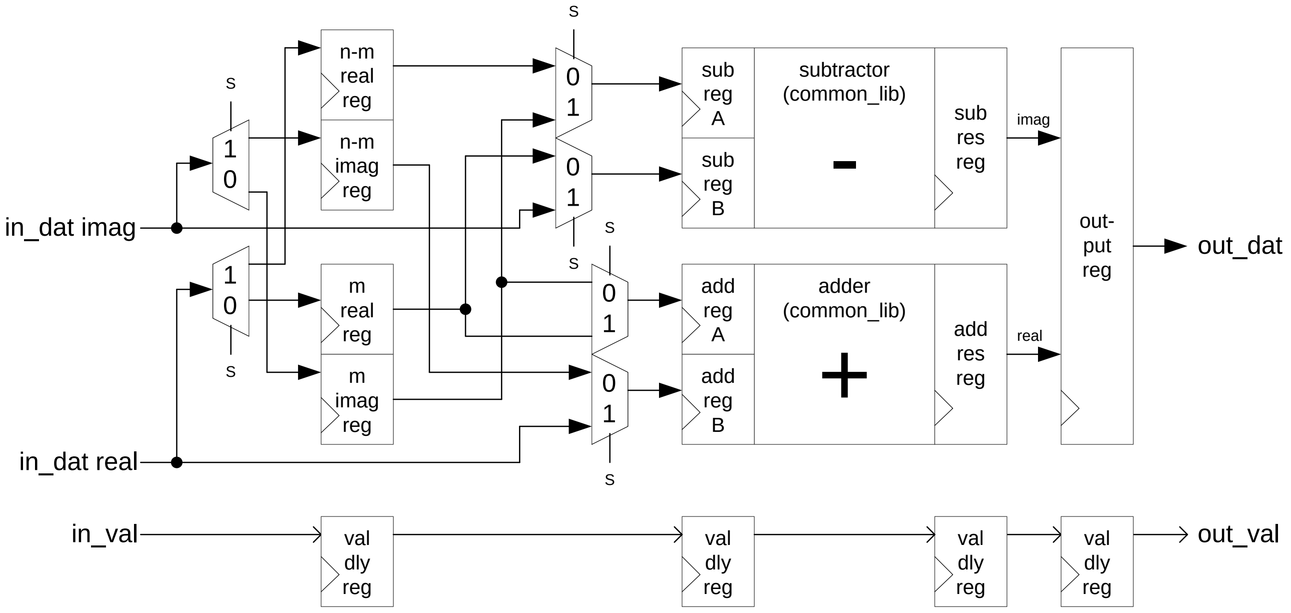


Figure 11 Block diagram of the fft_sepa unit

The result is a stream where the values of A and B are interleaved. The relation between the input signals, the output signals and the toggle signal S is given in Figure 12. Note the pipelined delay of 4 cycles that is introduced by the several registered stages.

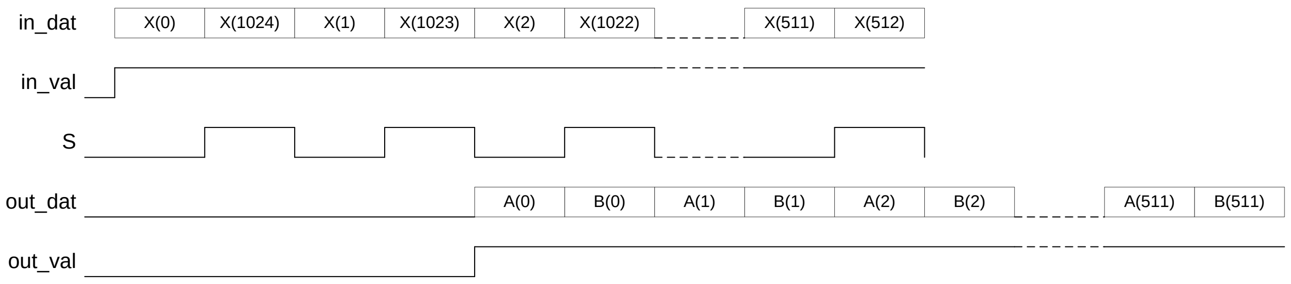


Figure 12 Timing diagram fft_sepa

5.2 fft_r2_par

The parallel variant of the FFT consists of the stages with parallel butterflies and the optional reorder and separate functions as shown in Figure 7 (which shows the instantiation of the 16-point variant. But of course it is possible to configure the FFT for every power of two). The number of butterflies is defined by N/2 * 2log(N).

5.2.1 fft_r2_bf_par

A dedicated butterfly, fft_r2_bf_par, for the parallel FFT is defined as shown in Figure 8. For the implementation several units from the rTwoSDF library are used. A detailed overview of the fft_r2_bf_par design is shown in Figure 13. The incoming data is first offered to two rTwoBF units (see 2.3.3 from [1] for detailed info). These units calculate the sum (c = a + b) and the difference (d = a – b) of their inputs. The sum-result is directly connected to the output buffer. The dif-result is send to the rTwoWMul unit where it gets multiplied with the appropriate twiddle factor. The output of the multiplier is connected to the output register.

5.2.1.1 Twiddle factor

The value of the twiddle factor is a function of the stage number in which the butterfly is instantiated and the butterfly's order number in that stage. Detailed information on the selection and creation of the twiddle factors can be found in 4.2 of [2].

5.2.1.2 Register delays

The fft_r2_bf_par design contains several registers to accommodate pipelined delays. The registers and their depth definition are listed in Table 9. The amount of delayed clock cycles is specified by the generics in the fft_pipeline record (see Table 4).

Register	Delay Parameters (g_fft_pipeline)
sum_reg	bf_lat + mul_lat
diff_reg	bf_lat
val_reg	bf_lat
out_reg	stage_lat

Table 9 Pipeline registers in fft_r2_bf_par

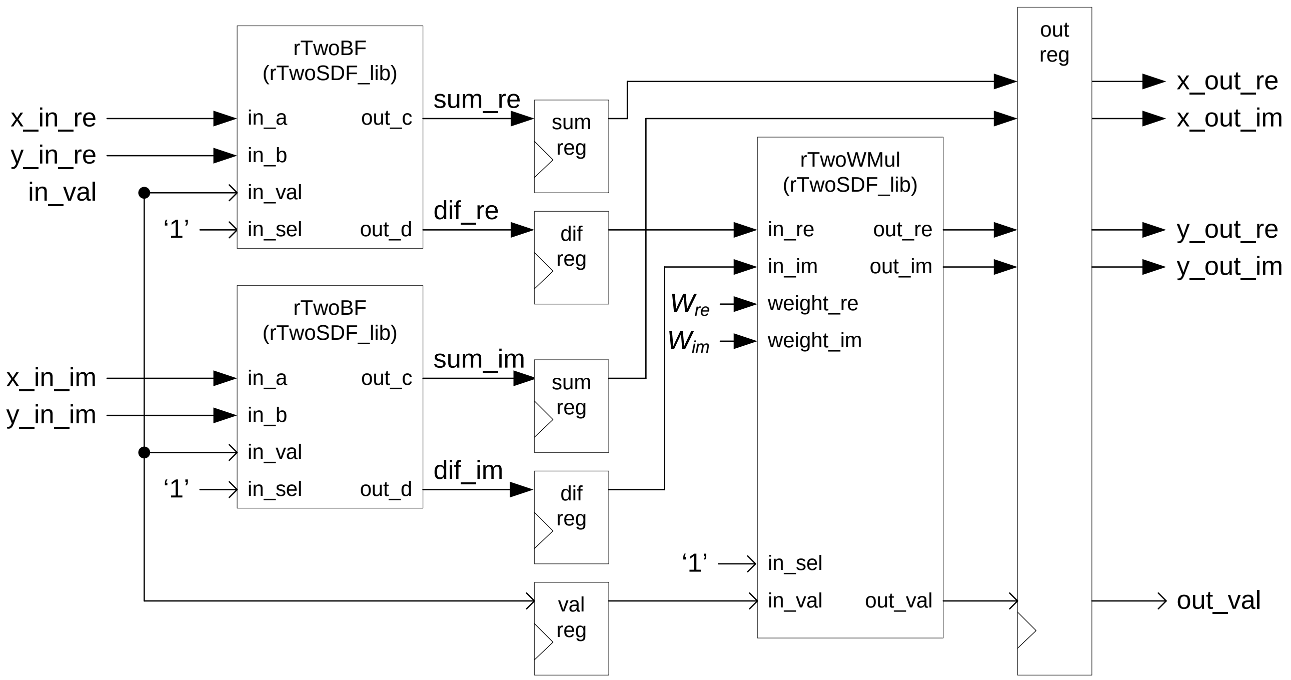


Figure 13 Block diagram of the fft_r2_bf_par unit

5.2.2 Connecting the parallel butterflies

The parallel stages are built of parallel butterfly units as described in section 5.2.1. Figure 14 shows the connection matrix for a 32-points parallel FFT. The parallel FFT contains 5 stages and each stage contains 16 elements. Every square represents a butterfly (fft_r2_bf_par) also referred to as element. The large number in each element represents the twiddle number. Now the output of each stage is connected to the input of the successive stage. This is done using an array of signals. Note that every element has two in- and outputs and therefor occupies two elements in the array. The small numbers near the output of each element show to which array indices (of the next stage) they are connected. For example:

- The second output of element 4 in stage 5 is connected to array index 24, which feeds the first input of element 12 in stage 4.
- The first output of element 11 in stage 3 is connected to array index 19, which feeds the second input of element 9 in stage 2.

A function (func_butterfly_connect) is created that determines the connection matrix between the butterflies in a generic way in order to make the parallel FFT configurable for every nof_points. The function uses the stage number, the output array index and the FFT's nof_points as arguments and it returns the array index to which the input port of the next butterfly is connected. Using this function to set the connectivity for the outputs of all elements will result in the complete connection matrix.

5.2.3 Parallel reordering

The implementation of the reordering consists of a wires only statement that uses the bit-flipped address to address the output array. Details can be found in chapter 4.4 of [5].

5.2.4 Parallel separation

Due to the parallel characteristics, the separate function of the pipelined FFT cannot be reused. The adders and sub tractors that are involved with the separation algorithm have to be instantiated in parallel. This means that for every output duo (X(m) and X(N-m)) a set of two adders(real + imag) and two sub tractors (real + imag) are required. Such a separation unit is shown in Figure 15.

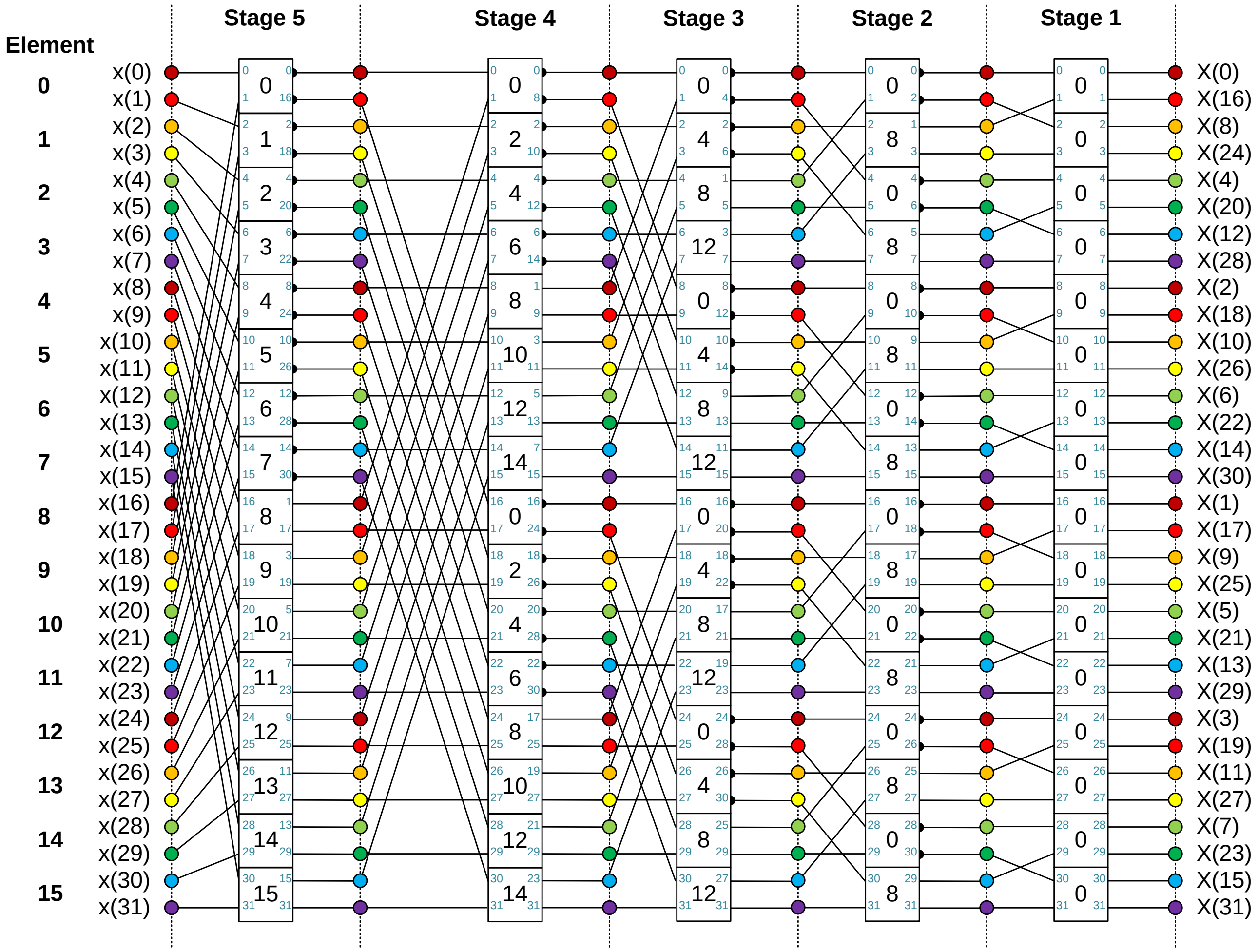


Figure 14 Connection matrix of a 32-points parallel FFT

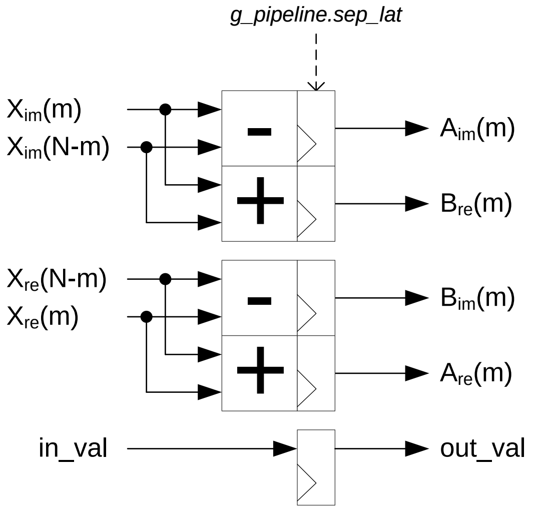


Figure 15 Parallel implementation of the separation algorithm

5.3 fft_r2_wide

Figure 9 shows the block diagram of a wideband FFT that is build out of multiple pipelined FFTs and one parallel FFT.

5.3.1 Generics

The parameters that are listed in Table 2 are used to configure the fft_r2_wide unit. The g_fft generics for the multiple fft_r2_pipe instances and the fft_r2_par are derived from the fft_r2_wide generics as shown in Table 10.

g_fft record field	**fft_r2_wide**	**fft_r2_pipe**	**fft_r2_par**
use_reorder	w_use_reorder	w_use_reorder	w_use_reorder
use_separate	w_use_separate	FALSE	FALSE
wb_factor	w_wb_factor	w_wb_factor	w_wb_factor
twiddle_offset	w_twiddle_offset	[0..w_wb_factor-1]	w_twiddle_offset
nof_points	w_nof_points	w_nof_points/w_wb_factor	w_wb_factor
in_dat_w	w_in_dat_w	w_stage_dat_w	w_stage_dat_w
out_dat_w	w_out_dat_w	w_stage_dat_w	w_stage_dat_w
stage_dat_w	w_stage_dat_w	w_stage_dat_w	w_stage_dat_w
guard_w	w_guard_w	0	w_guard_w
guard_enable	w_guard_enable	FALSE	FALSE
stat_data_w	w_stat_data_w	w_stat_data_w	w_stat_data_w
stat_data_sz	w_stat_data_sz	w_stat_data_sz	w_stat_data_sz

Table 10 g_fft generics derived

5.3.2 Modifications for rTwoWeights

The topological approach in section 2.5 of [1] shows that the twiddle factors (W_N^k) in the butterflies of all but the first pipelined FFTs require an offset ($W_N^{k+offset}$). In addition to this a 32-point wideband FFT with P=8 and P=4 are derived from the parallel FFT that is shown in Figure 14. The results are shown in Figure 16 and Figure 17 respectively. The rTwoWeights unit from the rTwoSDF library is modified to facilitate this offset feature. Two generics for the rTwoWeights unit are introduced to make this possible. These generics are only applicable for wideband configurations.

- g_twiddle_offset must be the pipelined FFT index in a wideband configuration
- g_stage_offset must be the log2(wb_factor)

Based on these generics and the value on the in_wAdr port the rTwoWeights unit determines the twiddle factor for the applicable butterfly. More about the indexing of the twiddle factors can be found in 4.2 of [2].

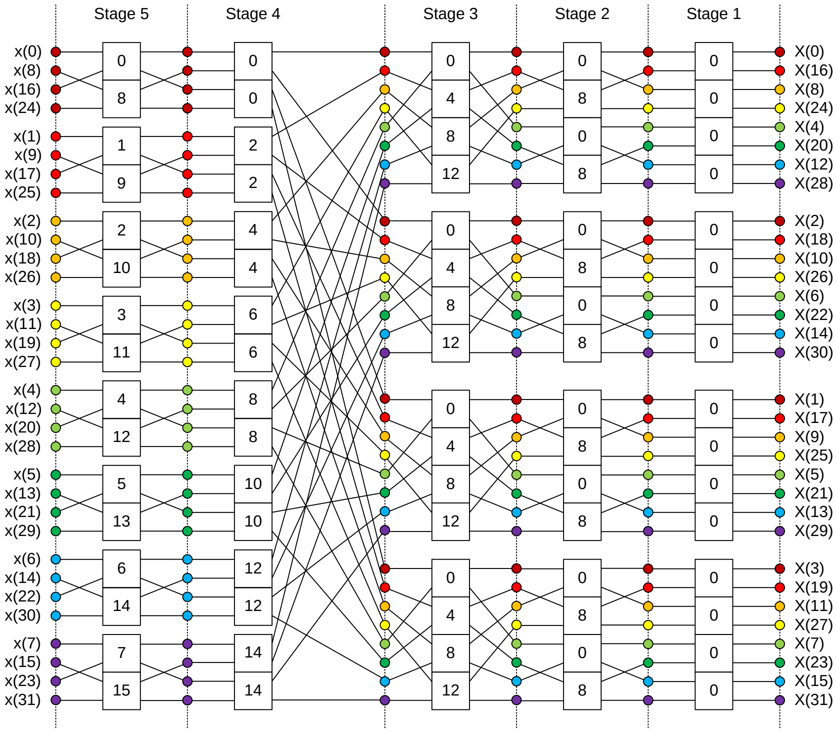


Figure 16 Butterfly block diagram for an N=32-point wideband FFT where P=8

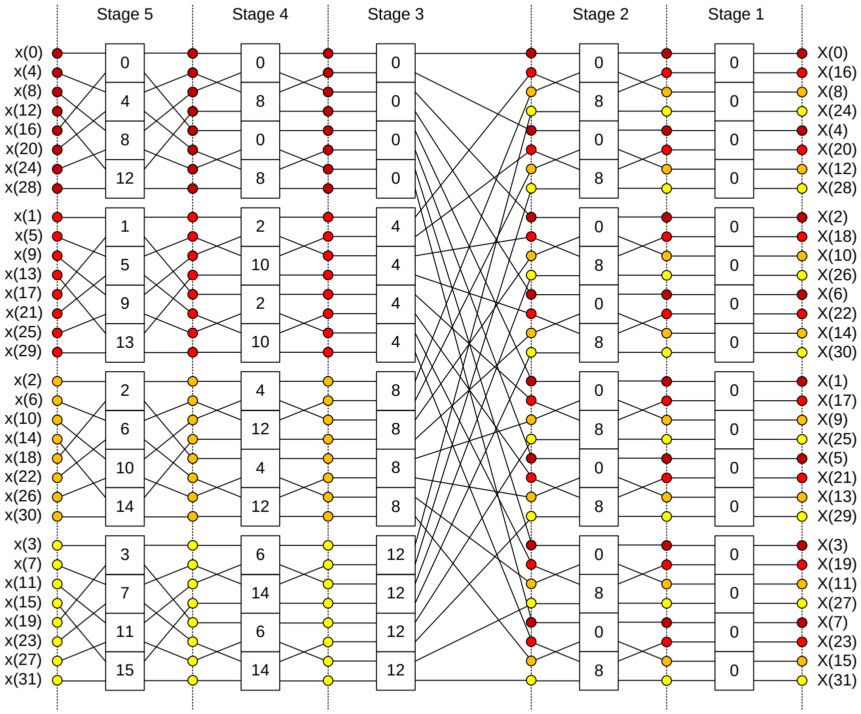


Figure 17 Butterfly block diagram for an N=32-point widenband FFT where P=4

5.3.3 fft_sepa_wide

The separation unit for the wideband FFT uses a dual paged ram for every P output of the wideband FFT. The streaming data is stored and once the first page is filled the unit will start reading the data from this first page. The configuration that is shown in Figure 17 (32-point wideband FFT, P=4) will be used as an example. The data of all 4 outputs is written to the first page of the dual paged rams. Table 11 shows the content of the rams after they are written. (Note that the re-ordering has taken place already).

RAM	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31

Table 11 32 frequency bins in 4 RAMs

In order to perform the separation algorithm data from different RAMs need to be combined and all RAMs need to be read at every clock cycle to keep up with the stream. This leads to the table of “combinations in time” depicted in Table 12 that shows the frequency bins in pairs. The frequency pairs are read out in parallel, but have to be serialized so they can be offered to an fft_sepa unit from the fft_lib (see paragraph 5.1.2.3 for more information). The serialization step is executed by the common_zip unit from the common_lib. The common_zip unit composes an output stream of multiple input streams, provided that the duty cycle of the in_val signal is 1/(nof_input_streams). The result of the common_zip units is offered to the fft_sepa units that will apply the actual separation algorithm. Figure 18 shows a detailed overview of the fft_sepa_wide unit, based on a wideband fft with a wb_factor of 4. The writing of the rams is straight forward. The in_val signal drives the write_enable and the next_page input is triggered when the address counter has reached the value of page_size-1. The read process is responsible for addressing the rams and the (de-)multiplexers. The required addresses for the rams are listed in the lower 4 rows of Table 12 (note that only two address strings are to be composed). The multiplexer and de-multiplexers both require a single bit address. Figure 19 shows the timing diagram for signals ‘S’, ‘F’ and the read addresses in relation to the next_page pulse. The read process uses an increasing and decreasing counter in combination with the switch signal ‘S’ to support the ram addressing.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
Stream 0	0		1		2		3	
	0(32)		31		30		29	
Stream 1		4		5		6		7
		28		27		26		25
Stream 2	8		9		10		11	
	24		23		22		21	
Stream 3		12		13		14		15
		20		19		18		17
adr ram 0	0x0	0x4	0x1	0x5	0x2	0x6	0x3	0x7
adr ram 1	0x0	0x4	0x1	0x5	0x2	0x6	0x3	0x7
adr ram 2	0x0	0x4	0x7	0x3	0x6	0x2	0x5	0x1
adr ram 3	0x0	0x4	0x7	0x3	0x6	0x2	0x5	0x1

Table 12 Output combinations in time

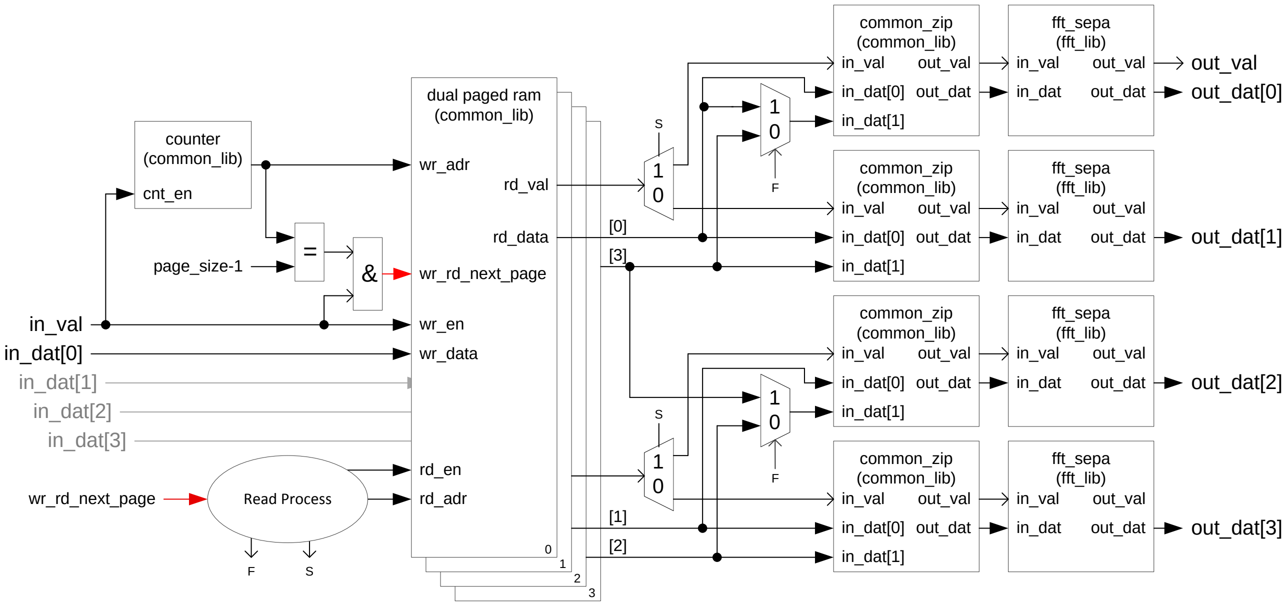


Figure 18 Detailed block diagram of the fft_sepa_wide unit (*P*=4)

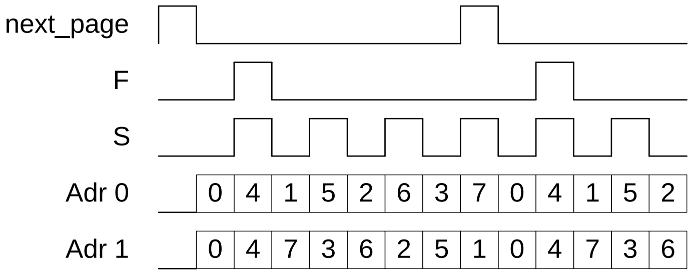


Figure 19 Timing diagram of fft_sepa_wide unit

5.4 Quantization

The quantization approach is for all types of FFTs (pipelined, wideband and parallel) the same and is called unconditional floating point scaling. It is based on scaling between every FFT butterfly stage in order to avoid clipping. In order to facilitate this quantization approach a number of generics are defined (see Table 3). The first step is performed at the input where the input data is resized to the stage_dat_w. The stage_dat_w is typical the width of the DSP multiplier (for a Stratix IV it is 18-bit). The second step consists of rescaling the data in between successive butterfly stages, which is typically a division by 2. In the last step the data of the last stage is re-quantized to meet the desired output width of the FFT. All quantization steps are performed with the requantize block from the common_lib. The requantize block are configured in such a way that they always use a rounding algorithm to determine the value of the MSB and LSB of the output. The following paragraphs explain these three steps and the relation to the generic parameters belonging to it in more detail.

5.4.1 FFT input resize

The initial resize at the input of the FFT takes into account the specified guard bits which are typical 2 bits since the stage-gain of the first two stages can be more than a factor of 2. See section 3.4 in [5] for more info. Figure 20 shows a graphical representation of the resize function where in_dat_w =8, stage_dat_w = 18 and guard_w = 2. The lower bits are filled with zeros. The result of this pre-scaling is that the added quantization errors have less influence on the eventual output.

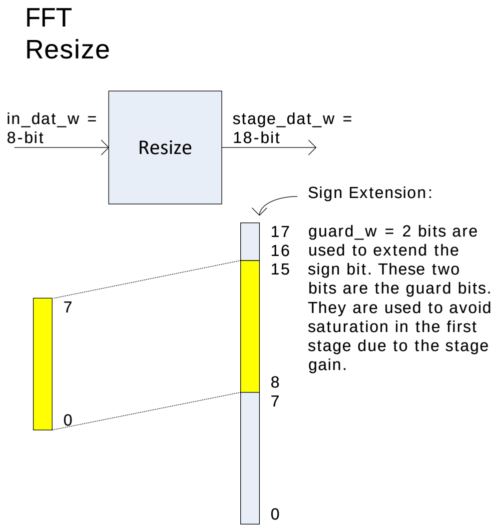

FFT
Resize
in_dat_w =
8-bit
Resize
stage_dat_w =
18-bit
Sign Extension:
guard_w = 2 bits are used to extend the sign bit. These two bits are the guard bits. They are used to avoid saturation in the first stage due to the stage gain.
7
0
17
16
15
8
7
0

Figure 20 FFT resize input data

5.4.2 FFT stage quantization

Every FFT stage has a potential gain-factor of 2 and therefore the data in between every stage is divided by two to avoid saturation in the datapath. The scaling is performed at the end of each stage in order to prepare the data for the following stage. A stage consists of a butterfly as depicted in Figure 21. In this example the stage_dat_w = 18 and the twiddle_w = 16. Note that the first quantizer(Q) in the d-output path compensates for the multiplication with the twiddle-factor where the second (and last) quantizer performs the stage quantization. Whether the stage quantization is performed or not at the end of a certain stage is determined by a generic: g_scale_enable. The reason for having the g_scale_enable generic is that the stage quantization should not be performed after the last stage, since there is not a next stage to protect from overflow. And because the FFT-input-resize guards with two bits, also the second to last stage does not require scaling as well.

5.4.3 FFT output quantization

The output of the last FFT stage is scaled to the desired outputs as specified with the out_dat_w generic as shown in Figure 22. In this example the 18-bit width output of the last stage is quantized to the required 14-bit width output. This is realized by taking away the lower 4 bits.

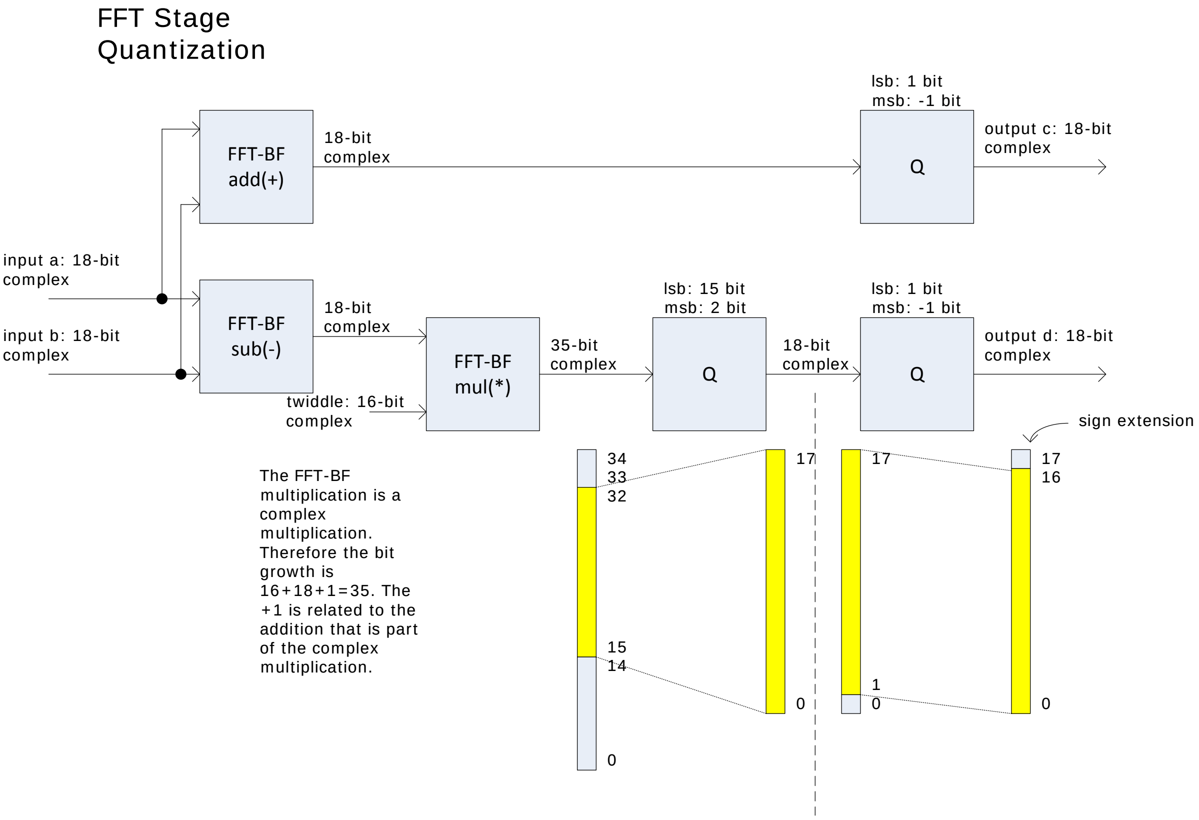


Figure 21 FFT stage quantization

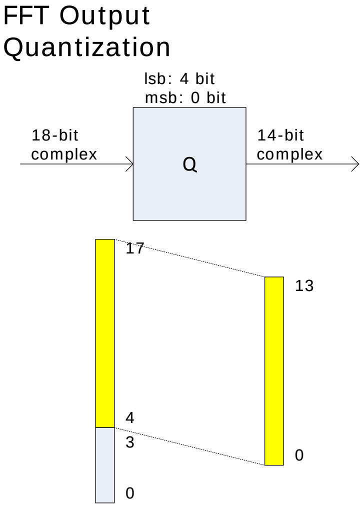


Figure 22 FFT output quantization

5.5 fft_wide_unit

The fft_wide_unit extends the fft_r2_wide design with the streaming SOSI and SISO interfaces as defined in [7] and with a statistics module which allows monitoring of the subband statistics. This is depicted in Figure 20. The fft_r2_wide is driven by the corresponding fields from the in_sosi_arr. The output of the fft_r2_wide is processed by the fft_wide_unit_control process that merges the fft data back into an array of SOSI streams that forms the out_sosi_arr. For every stream in the output array (out_sosi_arr) a quantizer-statistics pair is instantiated that calculates the subband statistics. The subband statistics can be read via the mm interface that combines the mm interfaces of all statistics units.

5.5.1 fft_wide_unit_control

The fft_wide_unit_control entity monitors the in_val signal that is driven by the out_val signal of the fft_r2_wide unit. Based on the assertion of the in_val signal it will compose the output SOSI streams that carry the frequency bins. Each packet in the output streams consists of nof_points/wb_factor frequency bins. Both the incoming SOSI fields BSN and ERR will be written to a FIFO. When the output streams are composed the BSN and ERR fields will be read from the FIFO’s. Incoming SYNC’s will be detected and the BSN that accompanies the sync will be stored. When the BSN that is read from the FIFO is the same as the stored one, the SYNC on the output will be asserted.

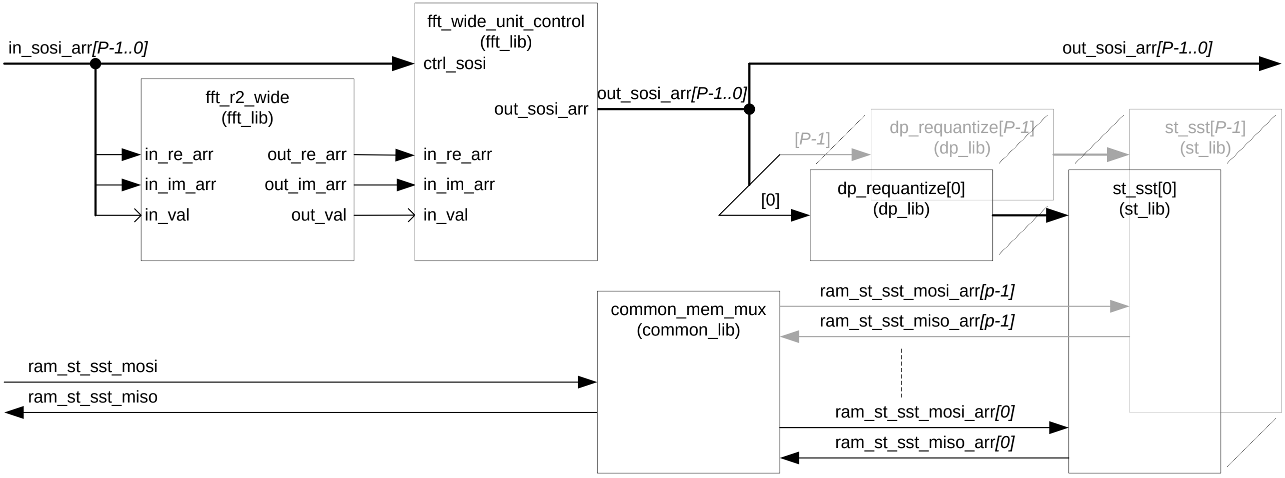


Figure 23 Detailed block diagram of fft_wide_unit

5.5.2 Quantizer

Currently the quantizer from the dp_lib is used to select 16-bit data out of the output streams, which is suitable for the subband statistics unit. The dp_requantize is the streaming equivalent of the requantize unit from the common_lib.

5.5.3 Subband Statistics

The subband statistics unit calculates the power of each subband and integrates this over a sync period. After each sync period the accumulated powers are written to a set of registers that can be read via the ram_st_sst_mosi interface. The st_sst unit comes from the st_lib, which was developed for the Lofar project. The source can be found in [8].

6 Verification

All testbench related files can be found in [9]. A dedicated package is created that holds procedures and definitions that are aimed to be reused in the testbenches. The package is called tb_fft_pkg.vhd.

6.1 tb_fft_r2_*

A general testbench approach is taken for the fft_r2_pipe, fft_r2_par and fft_r2_wide designs, which is depicted in Figure 21. The base for the testbench is inherited from the rTwoSDF_lib which is explained in detail in chapter 6 of [2]. The verification of the fft design is based in the usage of known input files and their accompanying golden reference files. The golden reference files have been verified using Matlab.
The testbenches are selftesting and can be run using the run –all command.

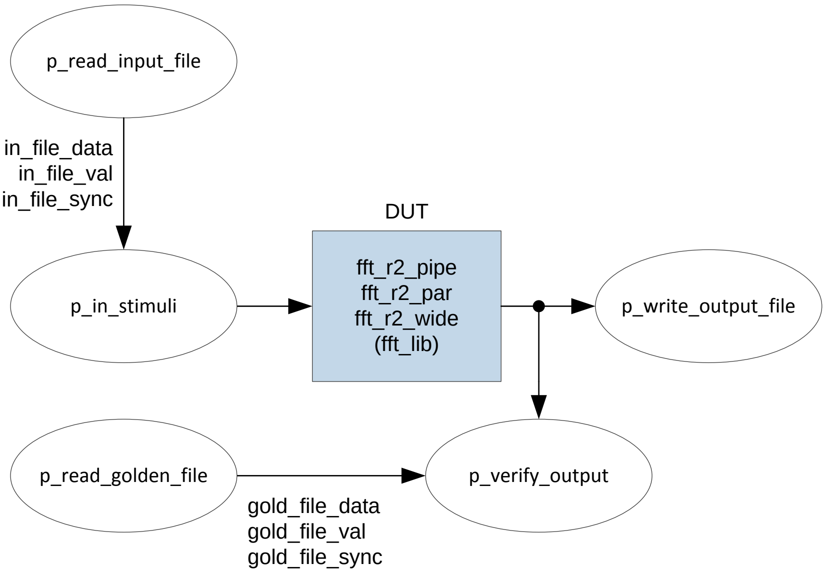


Figure 24 tb_fft_r2_* testbench

6.1.1 p_read_input_file

This process reads stimuli from a specified input file and places the data into the in_file_data, in_file_val and in_file_sync signals. The data source files are re-used from the rTwoSDF library and are located in [10].

6.1.2 p_in_stimuli

The p_in_stimuli process writes the input data (in_re, in_im, in_val and in_sync) to the DUT. The data is read from the signals that have been defined by the p_read_input_file process.

6.1.3 p_read_golden_file

This process reads the golden reference files that correspond to the chosen input file. The output data from this golden reference file has been verified with Matlab. The data is written to the signals gold_file_data, gold_file_val and gold_file_sync.

6.1.4 p_verify_output

The p_verify_output process reads the output data from the DUT and compares it with the reference data that is written to the gold_file_data, gold_file_val and gold_file_sync signals. An error message will be displayed in case the DUT output data and the reference data do not match.

6.1.5 p_write_output_file

During each simulation run the output of the DUT will also be written to an output file. This output file can be verified with Matlab and when the verification succeeds, the output file could be saved as a golden reference file.

6.2 tb_fft_wide_unit

The testbench for the fft_wide_unit is also based on the verification with the golden reference files as used in the rTwoSDF_lib. A block generator from the diag_lib is used to generate the input datastreams. Figure 22 gives an overview of the tb_fft_wide_unit.

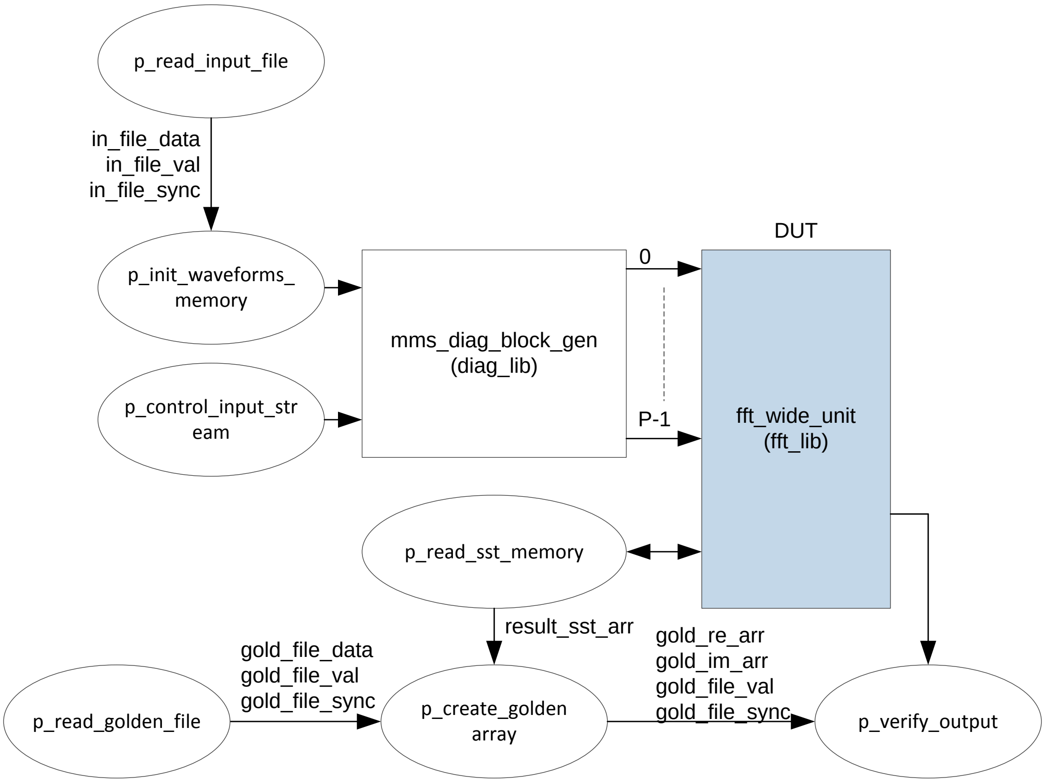


Figure 25 fft_wide_unit testbench

6.2.1 mms_diag_block_gen

In order to generate stimuli for the fft_wide_unit an mms_diag_block_gen instance is used. Although this is a RTL component, it can also be used for testbench simulation purposes. The block generator provides data on multiple streams that all comply with the SOSI format. Detailed information about the mms_diag_block gen component can be found in [11].

6.2.2 p_read_input_file

This process reads stimuli from a specified input file and places the data into the in_file_data, in_file_val and in_file_sync signals. The data source files are re-used from the rTwoSDF library and are located in [10].

6.2.3 p_init_waveforms_memory

This process writes the stimuli data to the waveform ram of the block generator at the start of the simulation.

6.2.4 p_control_input_stream

This process waits until the waveform rams have been written by the p_init_waveforms_memory process. Then it configures the block generator and starts the streams. It will also automatically stop the simulation when the specified amount of data has been send.

6.2.5 p_read_golden_file

This process reads the golden reference files that correspond to the chosen input file. The output data from this golden reference file has been verified with Matlab. The data is written to the signals gold_file_data, gold_file_val and gold_file_sync.

6.2.6 p_read_sst_memory

This process reads the integrated powers of the subbands: the subband statistics. The reading is triggered by a sync-pulse on one of the output streams of the DUT. The subband statistics are then read and written to the results_sst_arr signal.

6.2.7 p_create_golden_array

The p_create_golden_array re-arranges the gold_file_data signals into a gold_re_arr and a gold_im_arr that will be used by the p_verify_output process. It will also estimate the expected subband statistics and it verifies the subband statistics as well by comparing the expected statistics with the results_sst_arr.

6.2.8 p_verify_output

The p_verify_output process reads the output data from the DUT and compares it with the reference data that is written to the gold_re_arr and gold_im_arr signals. An error message will be displayed in case the DUT output data and the reference data do not match.

6.3 Pyhton based testbenches

Apart from the “traditional” testbenches as described in sections 6.1 and 6.2 there are also python based testbenches that can be used to verify the correct working of the FFT module. There is a python based testbench for the fft_r2_* units as well as for the fft_wide unit. The testbench architecture for both testcases is the same and is depicted in Figure 26.

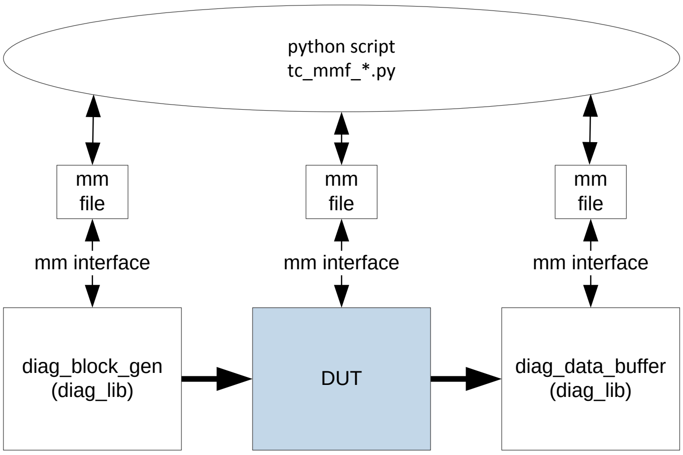


Figure 26 Architecture python based testbench

The VHDL part of the testbech consists of a diag_block_gen component that can provide one or more input streams to the DUT. The diag_block_gen is configurable via a mm interface. The output of the DUT is fed to one or more diag_data_buffers. The stored data can be retrieved from the data buffers also via a mm interface. All mm interfaces are connected to a mm file component that uses a file to respond to read and write requests initiated by the python script.
The python script is similar to a script that “talks” to hardware with the difference that the read and write requests are not send to hardware but to the mm files.

6.3.1 tb_mmf_fft_r2

The tb_mmf_fft_r2 testbench should be used in conjunction with the python script tc_mmf_fft_r2.py which can be found in the tb/python/ directory. The script allows to start Modelsim automatically, but it is also possible to start Modelsim, load and start the testbench manually. The script writes data to the diag_block_gen and based on the same data it generates the reference values to compare the output of DUT with.

6.3.2 tb_mmf_fft_wide_unit

Use tis testbench in conjunction with the tb/python/tc_mmf_fft_wide_unit.py script. The script works similar to the the tc_mmf_fft_r2.py script.

7 BN_FB Reference design

A reference design called bn_fb is made in order to test and validate the fft_wide unit in UniBoard hardware. This chapter describes this design and the peripherals that are used to perform the validation. Also the software and python-scripts that are used for validation are explained here.

7.1 Design

The bn_fb design consists of a SOPC system holding a NIOS II processor that connects to the ctrl_unb_common unit and the node_bn_fb unit via mm interfaces. The ctrl_unb_common unit contains basic peripherals like the ethernet interface, system info, I2C sensor access and a PLL for clock generation. The node_bn_fb unit is build out of two fft_wide_units and a complete instantiation of the node_bn_capture. The node_bn_capture unit provides 4 datastreams (sp_sosi_arr[3:0]). Each stream corresponds to an antenna input. The datastreams can carry data that is generated by the waveform generator or data directly from the ADC inputs. More information about the node_bn_capture design can be found in [6]. The mm interfaces for reading out the subband statistics of both fft_wide units are combined into one mm interface(ram_st_sst_mosi) using the common_mem_mux unit from the common_lib.

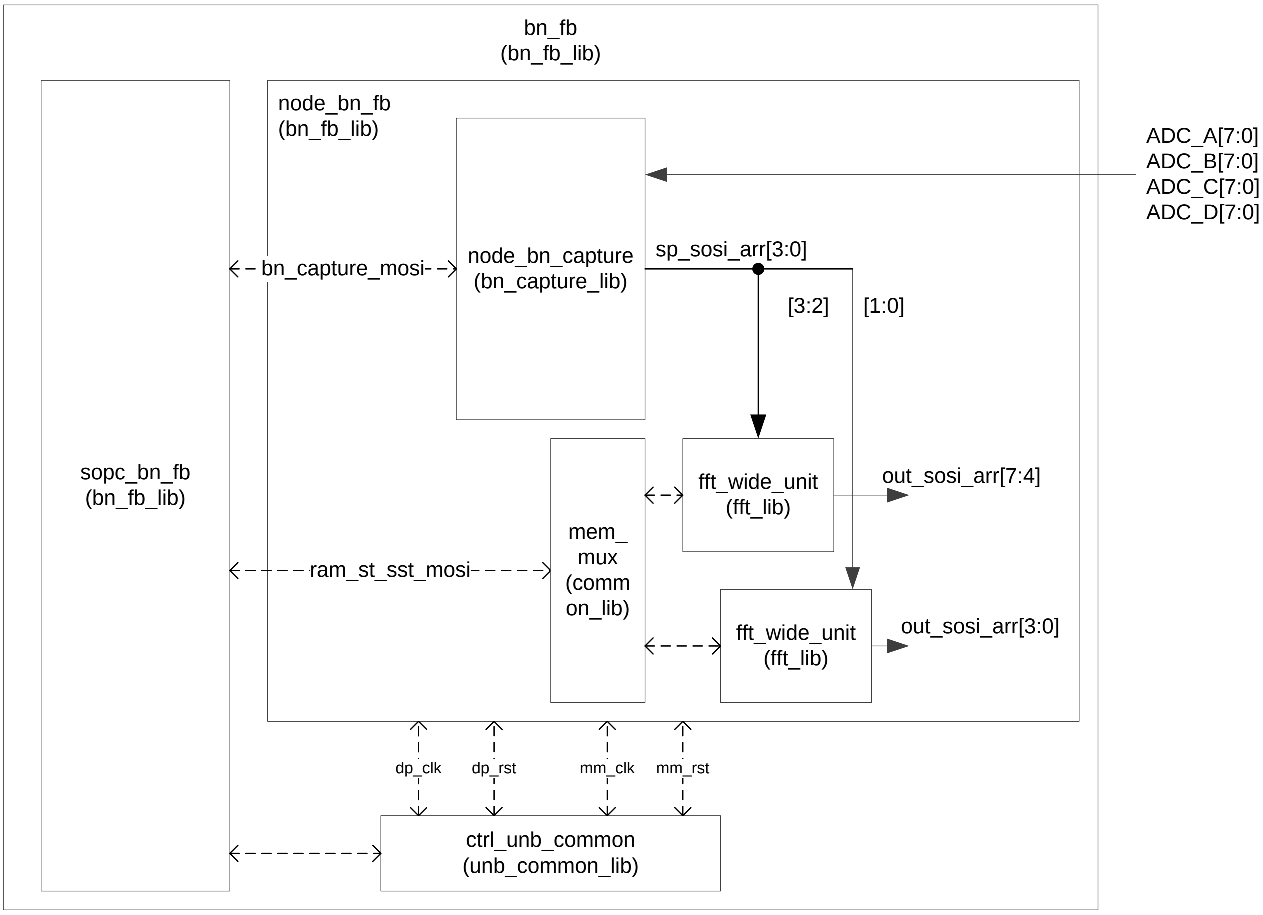


Figure 27: Design: bn_fb

7.2 Verification

The reference design is verified using a testbench: tb_node_bn_fb. An overview of the testbench is given in Figure 24. The testbench is used to manually check the correct working of the of the node_bn_fb design. Although the testbench is equipped with the verification process that is reused from the tb_fft_wide_unit (see 6.2) at the time of writing there are not yet golden reference files available for the node_bn_fb design. The tb_node_bn_fb testbench uses the waveform generator to generate input data for the FFTs. Different settings in terms of frequency, phase and amplitude for the waveform generator result in different outputs of the FFT. Therefore the correctness of the design is better examined in hardware, where the subband statistics can be plotted into a graph.

7.2.1 ctrl_unb_common

The ctrl_unb_common unit is part of the testbench in order to generate the dp_clk, dp_rst, mm_clk and mm_rst signals.

7.2.2 aduh_half

The aduh_half units from the aduh_lib simulate the conncted ADC that are on the ADU board. The testbench does not use these units actively. They are there to have the ADC-ports connected.

7.2.3 aphy_4g_800_mem_model

The aphy_4g_800_mem_model units represent the DDR3 memory modules. The memory models are used to have the DDR3 signals connected. The testbench does not apply stimuli that use the DDR3 functionality.

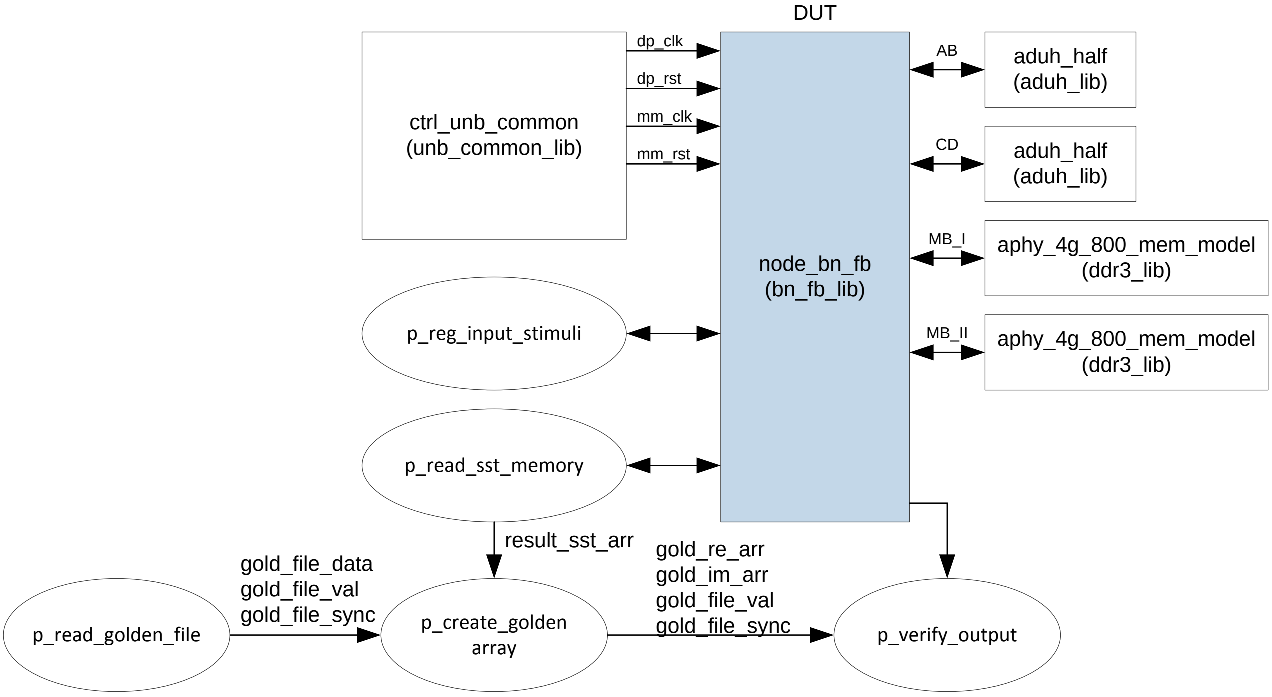


Figure 28 tb_node_bn_fb

8 Synthesis and Place & Route

TBD

9 Validation

Validation of the design is done using the reference design (bn_fb) in combination with a Python script that runs on a host PC. All validation is executed on a Uniboard.

9.1 Python

In order to validate the correct working of the fft_wide_unit a python testcase is created in the file tc_bn_fb.py. It is located in [12].

9.1.1 tc_bn_fb.py

The testcase allows enabling of the waveform generators or selection of the input data from the ADU's. The waveform generators can be configured to generate input signals of various frequencies. After setting up the input data the script reads out the subband statistics. The statistics are re-arranged and plotted. The user should then analyse of the plotted spectrum is as expected or not.

10 Appendix – list of files

10.1 Firmware VHDL

All VHDL source files that are used for the fft units can be found in the following two directories:

$UNB/Firmware/dsp/rTwoSDF/src/vhdl
$UNB/Firmware/dsp/fft/src/vhdl

The next table gives an overview of the VHDL source files:

VHDL File	Description
fft_pkg.vhd	Package that contains record en function definitions that are specific for the fft units.
fft_r2_par.vhd	Contains the design for a parallel fft.
fft_r2_pipe.vhd	Contains the design for a pipelined fft.
fft_r2_wide.vhd	Contains the design for a wideband fft.
fft_reorder_sepa_pipe.vhd	The reorder and separation function combined specific for the pipelined version of the fft.
fft_sepa.vhd	Low level unit that performs the separation algorithm, based on a streaming input and output.
fft_sepa_wide.vhd	Separation unit for the wideband fft.
fft_wide_unit.vhd	Design of a wideband fft that is extended with streaming input and output interfaces and a subband statistics module.
fft_wide_unit_control.vhd	Design unit that is part of the fft_wide_unit.

10.2 Testbench

The testbench files for simulation are in the following directory:

$UNB/Firmware/dsp/fft/tb/vhdl