ASTRON
Netherlands Institute for Radio Astronomy

APERTIF Filter Bank Firmware Specification

Part 2

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

Distribution list:

Group:	Others:
Andre Gunst Harm-Jan Pepping Daniel van der Schuur Raj Rajan Thilak	Gijs Schoonderbeek

Document history:

Revision	Date	Author	Modification / Change
0.1	2012-03-8	Eric Kooistra	First draft.

Table of contents:

Terminology:

ADC	Analogue to Digital Convertor
APERTIF	Aperture Tile In Focus
DFT	Discrete Fourier Transform
DIF	Decimate In Frequency
DIT	Decimate In Time
DUT	Device Under Test
Eop	End of packet
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HW	Hardware
IO	Input Output
IP	Intellectual Property
LOFAR	Low Frequency Array
LS	Least Significant (part, bit, byte)
MM	Memory-Mapped
MS	Most Significant (part, bit, byte)
PFB	Poly-phase Filter Bank
PFS	Pre Filter Structure for the PFT in a PFB
PFT	Pipelined FFT
R2SDF	Radix-2 Single Delay Feedback (pipelined FFT architecture)
RTL	Register Transfer Level
SISO	Source In Sink Out
Slice	FFT block size
SL	Standard Logic (VHDL)
SLV	Standard Logic Vector (VHDL)
SNR	Signal to Noise Ratio
Sop	Start of packet
SOSI	Source Out Sink In
ST	Streaming
SW	Software
Subband	Frequency bin of the PFB
UNB	Path to UniBoard Firmware directory [12]
WFFT	Wideband FFT
WPFS	Wideband PFS
WPFB	Wideband PFB

References:

1. “APERTIF Filter bank Firmware Specification”, feb 2011, ASTRON-RP-474, E. Kooistra
2. “Detailed Design of the Digital Beamformer System for APERTIF”, ASTRON-RP-413, G. Schoonderbeek, A. Gunst, E. Kooistra
3. “A R2SDF architecture based generic FFT for firmware implementation”, ASTRON-RP-755, dec 2011, R.T. Rajan
4. “A wideband FFT design for firmware implementation”, ASTRON-RP-880, dec 2011, R.T. Rajan
5. “Specification for module interfaces using VHDL records”, ASTRON-RP380, E. Kooistra
6. "RSP Firmware Design Description", LOFAR-ASTRON-SDD-018, Wessel Lubberhuizen, the relevant chapter is also available as LOFAR_pfb.pdf in $UNB/Firmware/modules/LOFAR/pft2/doc
7. “Brief description of the LOFAR station signal processing”, LOFAR-ASTRON-MEM-238, 2007, E. Kooistra
8. "Understanding Digital Signal Processing", R. Lyons
9. “Theory and Application of Digital Signal Processing”, L.R. Rabiner, B. Gold
10. “Polyphase filter bank quantization analysis”, 2004, LOFAR-ASTRON-MEM-109, J. Stemerdink
11. “Quantization Error Analysis of Digital Signal Processing Blocks”, 2004, LOFAR-ASTRON-MEM-129, J. Stemerdink
12. https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk, the UniBoard FP7 SVN repository ($UNB)
13. $UNB/Firmware/modules/LOFAR/pfs, contains the PFS
14. $UNB/Firmware/modules/LOFAR/pft2, contains the PFT and the PFB, doc/lofar_pfb_readme.txt
15. $UNB/Firmware/modules/dsp/rTwoSDF, contains rTwoSDF PFT, /doc/r2sdf_readme.txt
16. https://casper.berkeley.edu, CASPER community
17. “UniBoard Pulsar Project Definition”, April 2011, University of Manchester, Aziz AhmedSaid, e.a.
18. “A Scalable Correlator Architecture Based on Modular FPGA Hardware and Data Packetization”, Feb 4, 2008, Aaron Parsons e.a.
19. "A New Approach to Radio Astronomy Signal Processing: Packet Switched, FPGA-based, Upgradeable, Modular Hardware and Reusable, Platform-Independent Signal Processing Libraries", 200509URSI.pdf, Aaron Parsons e.a.
20. “The Symmetic Group in Data Permutation, with Applications to a High-Bandwidth Streaming FFT Architecture”, Aaron Parsons
21. www.rfel.com

1 Introduction

1.1 Purpose

This document specifies the Wideband Poly-phase Filter Bank (WPFB) firmware module that is required for APERTIF [2]. This document is called 'part 2' because it supersedes the previous specification document [1]. In addition this document also provides extra information to facilitate the development of the WPFB.

1.2 Background

The result of the previous specification [1] was the R2SDF pipelined FFT VHDL firmware implementation described in [3], plus a design note on how to extend the FFT to a wideband implementation for higher sample rates [4]. The R2SDF pipelined FFT will be reused in the wideband filterbank.

The wideband filterbank is a poly-phase filterbank (PFB). The term wideband is used to indicate that the data sample clock rate is a factor P higher than the digital processing clock rate. The wideband factor $P \geq 1$ and typically P is a power of 2. The LOFAR PFB module [6] was ported to UniBoard and is available at [13, 14], but only supports $P = 1$. For APERTIF $P = 4$ is needed to be able to process 800 MSps using a digital clock frequency of 200 MHz [2]. The wideband filterbank consists of:

- A wideband Finite Impulse Response (FIR) pre-filter structure (WPFS)
- A wideband pipelined complex Fast Fourier Transform (WPFT) including a separate function to be able to use a complex FFT for two real inputs

1.3 Planning

The complex pipelined FFT is the R2SDF pipelined FFT available at [15] and forms the building block for the wideband filterbank. Hence the WPFB will not reuse the LOFAR PFT. The next development steps are:

- Parallel FFT
- Wideband complex FFT
- Wideband complex FFT for two real inputs
- Wideband PFS
- Wideband PFB for two real inputs

The SOSI sync, sop, and eop signals are not essential for the operation of the WPFB. Therefore these IO signals should be added to the WPFB as a final step after the functional development of the WPFB and its internal components has been done.

2 Functional specification

2.1 General

The Wideband Poly-phase Filter Bank is referred to as WPFB.

- The WPFB firmware should be implemented in a generic and modular way.
- The WPFB implementation must support *P* power of 2 and gracefully include *P*=1.
- The WPFB performance only needs to be verified for the settings of the APERTIF beamformer [2] application case, so for 512 subbands (so an *N*=1024 points FFT), 8-bit real input samples, 800 MSps and 200 MHz digital processing clock (so *P*=4).
- The WPFB must be capable of handling arbitrary data invalid gaps in the input data stream. Being able to cope with data invalid helps to ensure a proper implementation of the WPFB.

2.2 Using standard logic vectors or sosi records

For the wideband signals the data of *P* samples arrives in parallel. The choice is to:

1. Use *P* separate standard logic vector (slv) signals in parallel
2. Concatenate the *P* signals into a single slv
3. Use a sufficiently wide general slv array like e.g. t_rtwo_slv_arr in rtwo_pkg.vhd (subtype of t_slv_32_arr in $UNB - common_pkg.vhd) to contain the *P* signals
4. Use a single sosi record (from $UNB – dp_stream_pkg.vhd [5]) for all *P* data samples by concatenating them in the slv data field or in the ‘re’ and ‘im’ fields
5. Use an array of *P* sosi records (from $UNB – dp_stream_pkg.vhd [5])

Option 1 is suitable for *P*=1 but it discarded for *P*>1, because the entity IO then fixed so not flexible for supporting different values for *P*. Option 2 and 4 are discarded because concatenating data signals into a single slv is awkward to read as hexadecimal numbers when the data width in not an integer number of nibbles. Hence this leaves using option 3 with an array of *P* sufficiently wide slv or option 5 with an array of sosi records. Both are fine. The disadvantage of using sosi records is that many fields are unused and in case of large arrays the relevant fields get more difficult to view in the Wave Window. Therefore option 3 is deemed most suitable to use for the WPFB and its components.

In addition it option 5 can still be supported by defining wrapper entities that map the IO to the sosi records. This may typically be useful for the higher level blocks like the WPFS, WFFT and the WPFB.

2.3 Complex FFT

2.3.1 Algorithm

The FFT consists of butterfly operations. Figure 1 shows the optimized structure of the FFT butterfly for decimation-in-frequency as explained in section 4.6 in [8] (left) and an instance diagram (right).

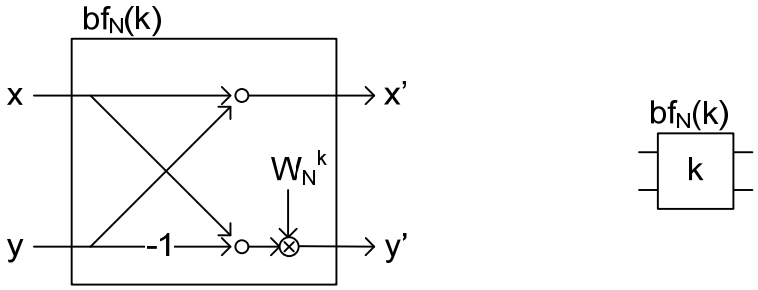

Figure 1: Optimized form of butterfly structure for decimation-in-frequency

The term $W_N=e^{-j2\pi/N}$ and is called the twiddle factor. Note that:

- The choice between using decimation-in-frequency (DIF) or decimation-in-time (DIT) is arbitrary. The rTwoSDF [3] uses DIF.
- Using DIF or DIT has nothing to do with whether the FFT output will be bit-reversed or not, see section 4.5 in [8]. The bit reversing is due to the crossing signal flows that are needed to let each input sample be weighted directly or indirectly by the twiddle factors (carrier waves).

Figure 2 shows the signal flow diagram for a parallel N=16 point FFT using DIF and with-bit reversed outputs. The twiddle factors in Figure 2 are represented by the exponent k of W_{16} and follow from section 13.16 in [8].

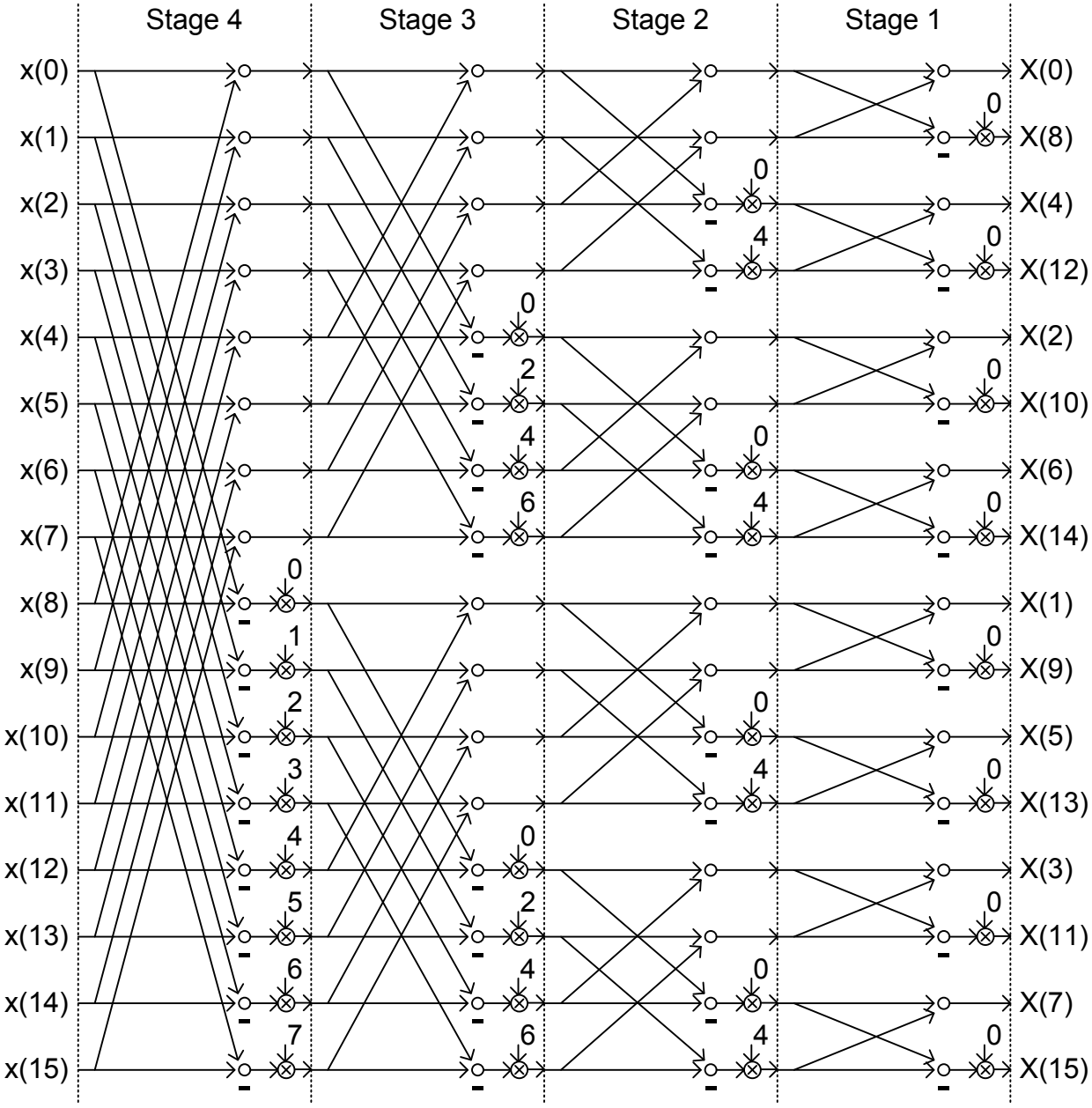

Figure 2: Signal flow diagram for a parallel N=16-point decimation-in-frequency FFT

2.3.2 rTwoSDF PFT entity

Figure 3 show a block diagram of the Pipelined complex FFT (PFT) that is implemented in rTwoSDF.vhd [3].

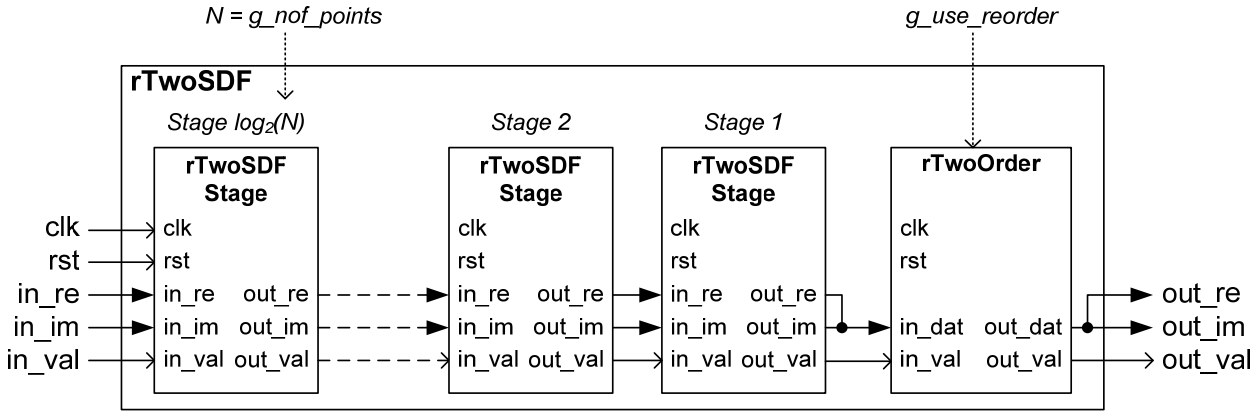

Figure 3: Block diagram of the rTwoSDF implementation in rTwoSDF.vhd

The stage numbers in Figure 3 and rTwoSDF.vhd correspond to the stage numbers in Figure 2. The rTwoSDF PFT has two modes:

1. *g_use_reorder* = false yields bit reversed order output
2. *g_use_reorder* = true yields normal order output

The tb_rTwoSDF.vhd uses *g_use_reorder* =true to verify the PFT using a complex noise input signal with a know result.

2.3.3 rTwoSDF PFT architecture

The pipelined aspect of the PFT appears from the fact that each stage processes its *N* inputs sequentially in time. Hence each stage only needs the hardware resources for a single butterfy operation of Figure 1. It is important to distinguish the algorithm pipelining of z^{-1} sample delays from the digital pipelining for RTL clock delays. In Figure 2 the algorithm pipelining progresses vertically from top to bottom. The RTL pipelining that is necessary to achieve timing closure for higher processing clock rates can be placed horizontally between the stages. The rTwoSDFStage has several internal pipeline settings for z^{-1} delays and for RTL delays that are kept in a record in rTwoSDFPkg.vhd. The rTwoSDF simulates OK, also when all RTL pipelining delays are set to 0 and also when in_val is only active at arbitrary clock cycles. Figure 4 shows the block diagram of rTwoSDFStage.

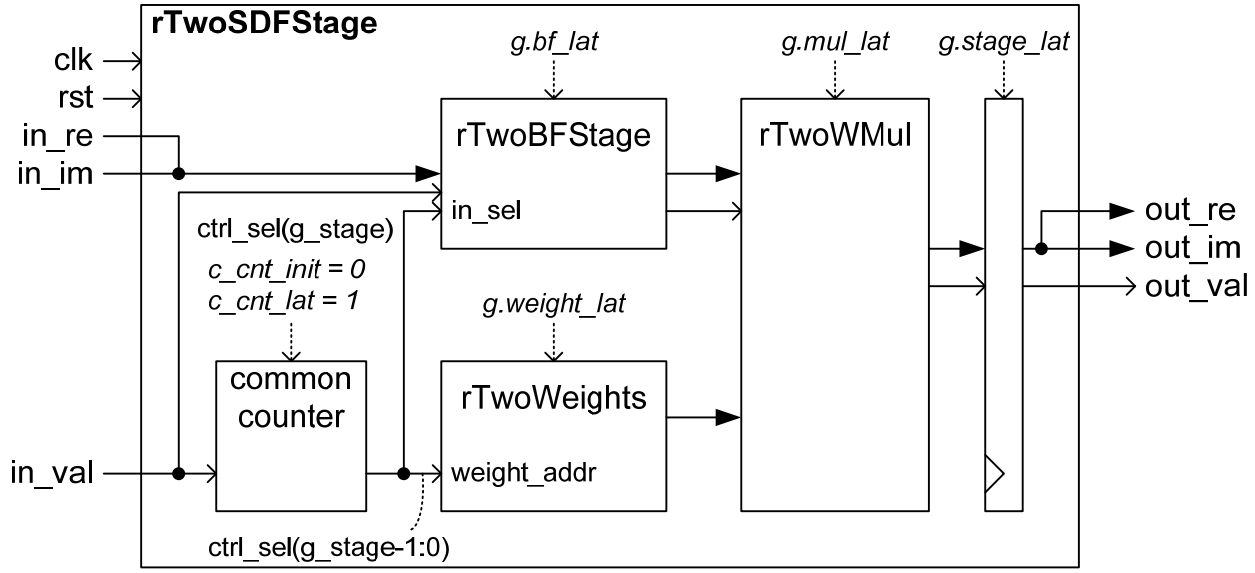

Figure 4: Block diagram of rTwoSDFStage.vhd

The feedback delay lines (FIFOs) to store the data for the z^{-1} delays are located inside rTwoBFStage as shown in Figure 5. Note the AND-gate to make out_val. After the stage delay the feedback valid output goes high and remains high and acts as an enable for in_val to out_val. This is important for controlling the local counter in each rTwoSDFStage that controls the BF select and the twiddle selection (see Figure 4).

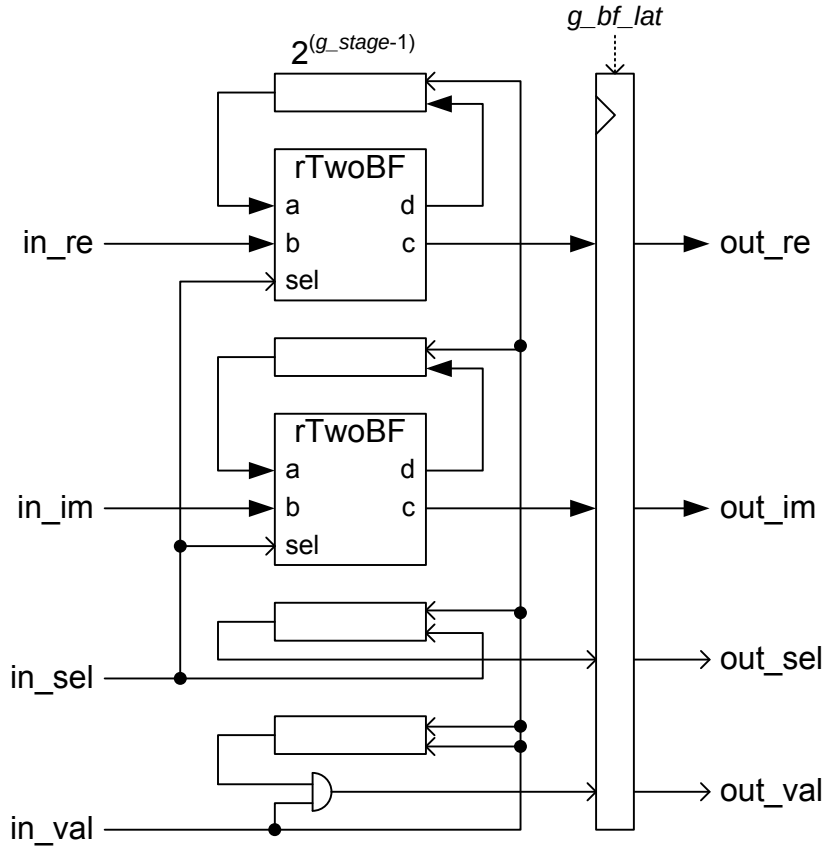

Figure 5: Block diagram of rTwoBFStage with the feedback delay lines

The rTwoBF in Figure 5 implements:

```
c = a + b when sel='1' else a
d = a - b when sel='1' else b
```

There is the option via t_rtwo_sdf_stage_pipeline to move some of the feedback z^{-1} delays into rTwoBF to ease timing closure for synthesis, but typically this is not useful.

Dependent on the in_sel input the rTwoWMul in Figure 4 multiplies the rTwoBFStage output with the twiddle factor from rTwoWeight or it passes the rTwoBFStage output on.

The rTwoWMul also performs truncation and resizing on the product. For safe scaling the output of each FFT stage must have 1 bit extra than the input [10, 11]. The twiddle factors are normalized 16 bit numbers, therefore the 16 - 1 = 15 LSbits of the product get truncated. The truncate function keeps the MS part. The remaining product vector is then resized to the output width. The resize function extends the sign bit or keeps LS part. Note that the resize function differs from the resize function in the IEEE numeric_std VHDL package, because that resize function always preserves the sign bit. All rTwoSDFStage stages use the same data width as the final rTwoSDF g_out_dat_w output width. By choosing g_out_dat_w = g_in_dat_w + ½(1+log2(*N*)) the contribution of the FFT implementation to the quatization noise is about equal to the input quantization noise [11]. Therefore the tb_rTwoSDF.vhd uses 6 bits extra for the output when g_nof_points = *N* = 1024.

For more information on the implementation of the rTwoSDF PFT see [3] and the r2sdf_readme.txt in [15].

2.4 Complex FFT for two real inputs

Figure 6 shows the Pipelined FFT (PFT) for a time series complex input, including the option to use two real inputs.

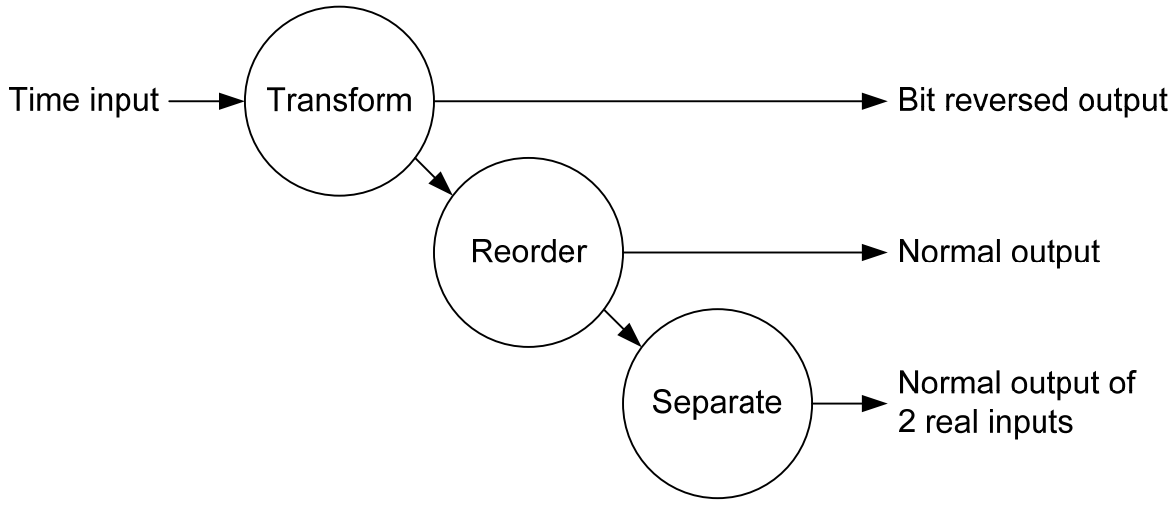

Figure 6: Data flow diagram for the PFT

The PFT has 3 modes:

1. bit reversed order output
2. normal order output
3. normal order output separated for 2 real inputs

Figure 7 shows the interface signals for the PFT at the input ($x = x_{re} + jx_{im}$) and after the 'transform' with 'reorder' ($X = X_{re} + jX_{im}$) and after 'separate' ($a = x_{re}$, $b = x_{im}$, $A = A_{re} + jA_{im}$ and $B = B_{re} + jB_{im}$).

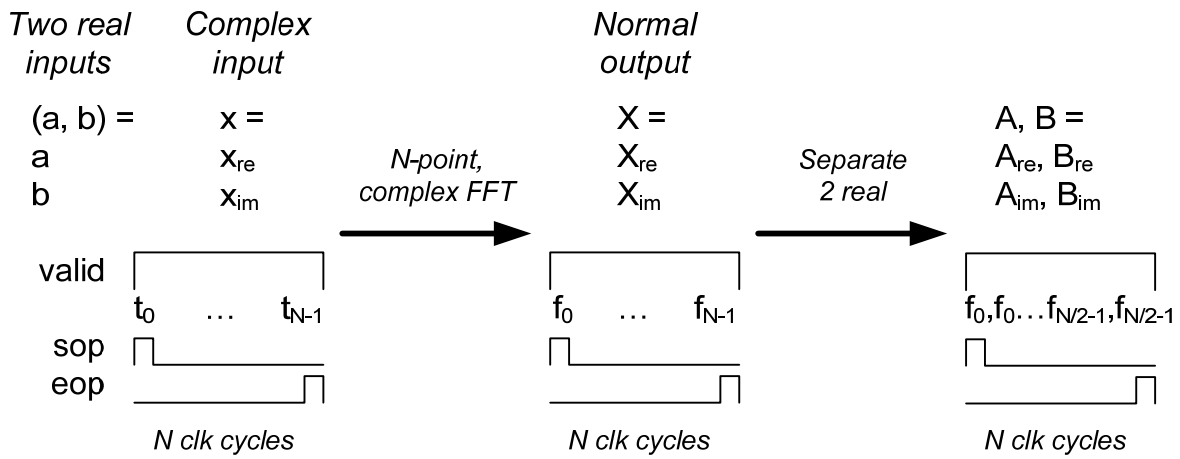

Figure 7: Interface signals for the PFT

The separated outputs *A* and *B* are interleaved as shown in Figure 7 with their real and imaginary parts available in parallel at the same time. A block of *N* real input samples for *a* and for *b* results in a block of *N*/2 complex frequency samples for *A* and for *B*. These complex frequency samples for *A* and for *B* are called subbands [7].

2.4.1 Reorder function

The reorder function is already available in rTwoOrder.vhd. This rTwoOrder uses a dual page memory whereby *N* samples are written in one page while the previous *N* samples are read from the other page. Reference [20] explains that at the expense of some more addressing logic it is possible to use a single page memory to achieve the reordering. However for our purposes the existing rTwoOrder.vhd suffices so no improvements should be made for that yet.

2.4.2 Separate function

The separate function is available in the LOFAR PFT pft_separate.vhd, but it is better to redesign it for the APERTIF PFT and call it rtwo_separate.vhd. The separate function follows from section 13.5 in [8] and is defined by:

$$2\,A_{re}(m) = X_{re}(N-m) + X_{re}(m)$$
$$2\,B_{im}(m) = X_{re}(N-m) - X_{re}(m)$$
$$2\,A_{im}(m) = X_{im}(m) - X_{im}(N-m)$$
$$2\,B_{re}(m) = X_{im}(m) + X_{im}(N-m)$$

For m = 0 to N/2-1, whereby $X(N)=X(0)$, which is automatically achieved by using modulo N addressing. The FFT output *A* is conjugate symmetric so only the first N/2 values need to be calculated, similar for *B*.

2.4.3 PFT entity and interface

The PFT entity is called rtwo_pft.vhd. Conform Figure 6 rtwo_pft.vhd instantiates rTwoSDF.vhd,vhd from section 2.3 that implements the 'transform' and 'reorder' and it instantiates rtwo_separate.vhd from section 2.4.2 that implements the 'separate'. The interface for rtwo_pft is defined in Table 1.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path resetReal input
in_re	SLV	IN	in_dat_w	Real input x_{re} or a
in_im	SLV	IN	in_dat_w	Imaginary input x_{im} or b
in_valid	SL	IN	-	Input data valid strobe
in_sop	SL	IN	-	Input start of data block
in_eop	SL	IN	-	Input end of data block
out_re	SLV	OUT	out_dat_w	Real output X_{re} or A_{re},B_{re}
out_im	SLV	OUT	out_dat_w	Imaginary output X_{im} or A_{im},B_{im}
out_valid	SL	OUT	-	Output data valid strobe
out_sop	SL	OUT	-	Output start of data block
out_eop	SL	OUT	-	Output end of data block

Table 1: Interface signals for rtwo_pft

The sop and eop mark the start and end of the FFT block (also called FFT slice) as shown in Figure 7. The rtwo_pft should merely delay them appropriately. The other sosi fields are not used and the PFT does not support backpressure so there are no siso signals. The rtwo_pft should use a single generic record t_rtwo_fft as defined in Table 2. For the rTwoSDF pipeline settings the default c_rtwo_sdf_stage_pipeline settings from rTwoSDFPkg can be used (section 2.3).

Generic field	Type	Default	Description
use_reorder	Boolean	true	When false output in bit reversed order, else output in normal order
use_separate	Boolean	true	When false output for complex input, else separates the output for two real inputs
wb_factor	Natural	1	Wideband factor *P* is not used in rtwo_pft so effectively *P* = 1
twiddle_offset	Natural	0	Twiddle offset for PFT sections in a WFFT, default 0 for *P* = 1.
nof_points	Natural	1024	FFT size *N*
in_dat_w	Natural	8	Input data width
out_dat_w	Natural	13	Output data width

Table 2: Generic record type t_rtwo_fft for PFT and WFFT

The record type and default value c_rtwo_pft should be kept in the package rtwo_pkg.vhd. If necessary this package can also include definitions for the other components within the rtwo_pft.

2.4.4 Biplex PFT

Examining Figure 2 it becomes clear that the multipliers in the rTwoSDF are only used 50% of the time. A biplex PFT gets this to 100% usage by applying a second input stream to the PFT in such a way that this stream can use the multiplier the other 50% of the time [18, 19, 20]. However for our purposes the existing rTwoSDF.vhd suffices so no 'biplex' improvements should be made for that yet.

2.5 Wideband complex FFT

2.5.1 Topological approach

Figure 8 shows a principle block diagram for the Wideband FFT that was derived from figure 3 of [1] that was reproduced from reference [21].

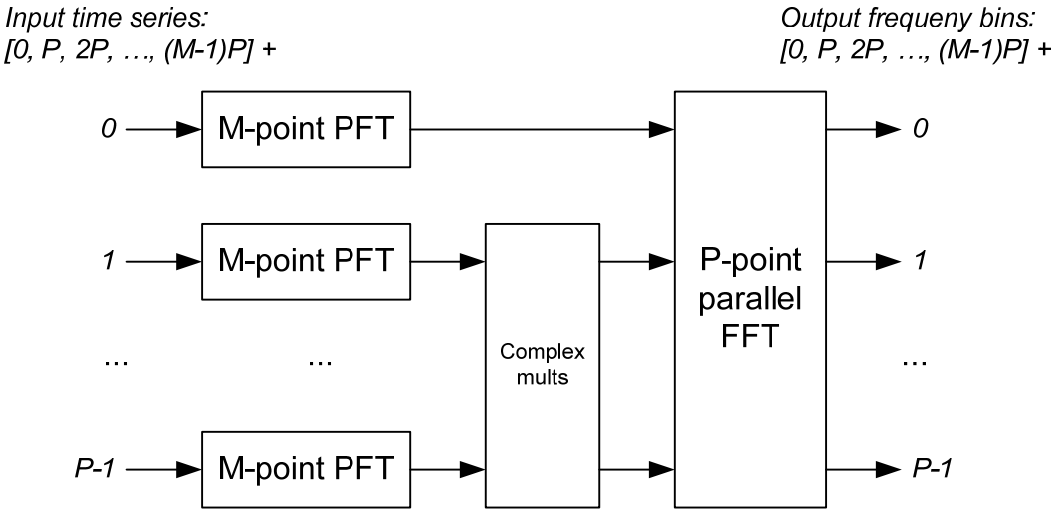

Figure 8: Principle block diagram of the Wideband FFT based on [21] reproduced in [1]

In principle an *N*-point WFFT consists of *P M*-point PFT sections followed by a *P*-point parallel DFT, whereby *N=P*M*. Note that the parallel DFT becomes an parallel FFT for *P* powers of 2. Between the PFT sections and the parallel DFT the outputs of the PFT sections get rotated dependent on the section number. The CASPER community [16] have a WFFT and WPFB, but there is no reference that describes these in detail [18, 19], although figure 3 in reference [20] provides some more insight. The WFFT could be seen as the problem of performing a large FFT using small FFTs, this problem is analyzed section 13.47 in [8] and in [17]. In [4] Raj Rajan Thilak derives a structure for a WFFT and provides a MATLAB simulation for it. However this structure does not seem to solve the parallel DFT as an FFT. Most these references take a mathematical approach. An alternative is to take a topological approach based on the data flow graph of the parallel FFT in Figure 2. Figure 9 shows the same as Figure 2 but now using the more compact butterfly representation from Figure 1 with $bf_N(k)$ indicated by *k*.

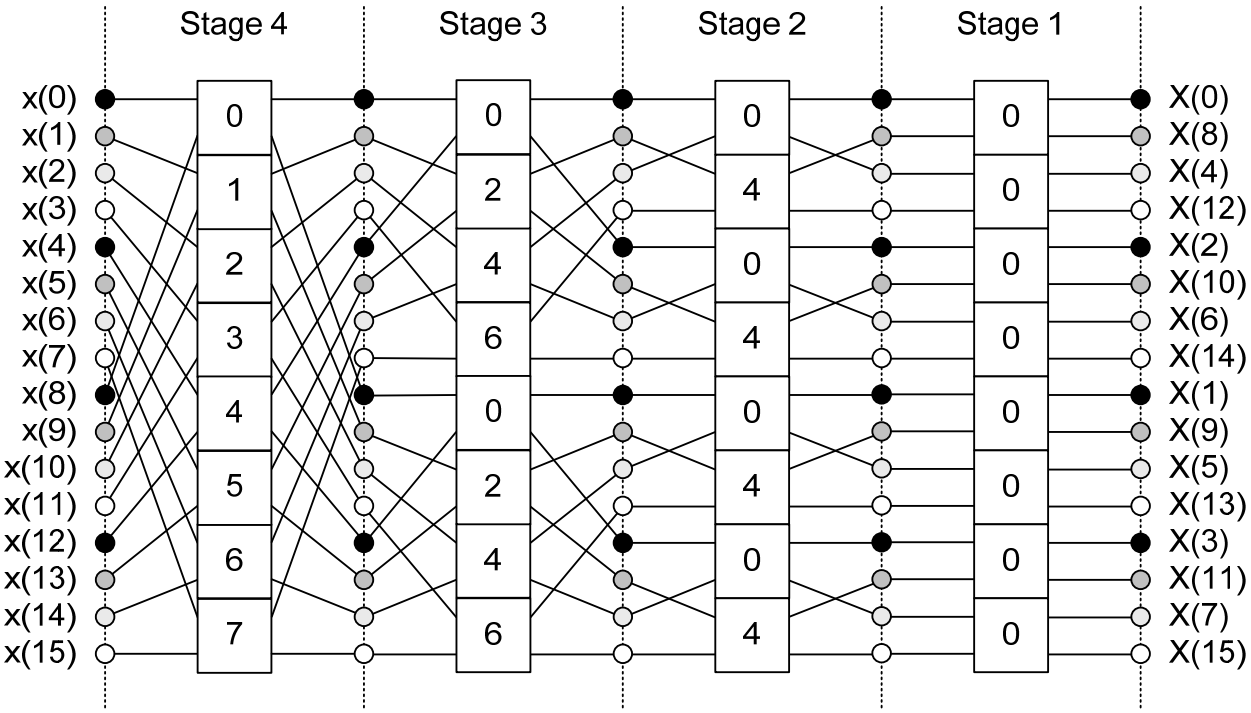

Figure 9: Butterfly block diagram for a parallel N=16-point decimation-in-frequency FFT

Similar as in figure 3 in reference [20] the samples in Figure 9 with the same color appear serially, while samples of different color arrive in parallel. For P=4 stage 1 and 2 = $\log_2(P)$ need to be processed in parallel while the other stages can still be pipeline processed. The topological approach is now to redraw Figure 9 as shown in Figure 10.

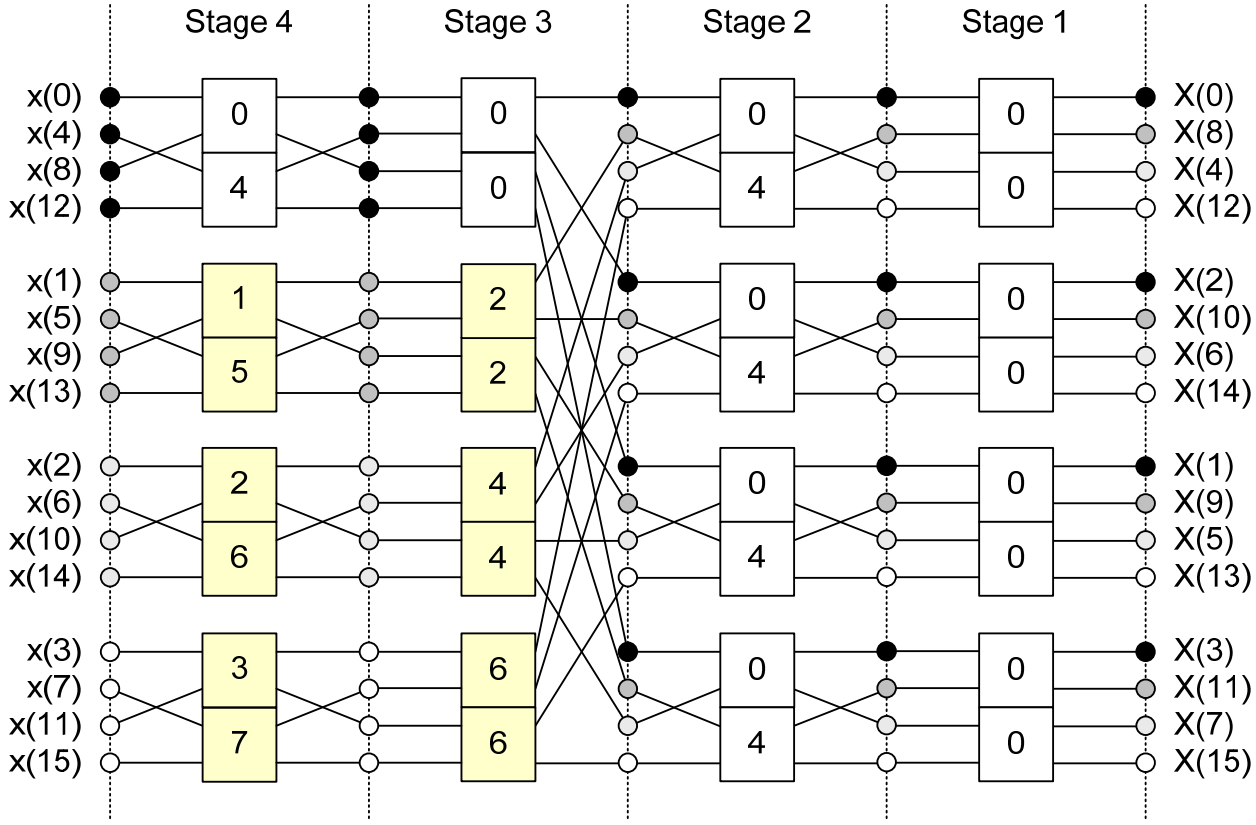

Figure 10: Butterfly block diagram for an N=16-point wideband FFT

Clearly the wideband FFT consists of P M-point PFT sections and P P-point FFT sections. The PFT sections need an additional twiddle_offset parameter to be able to adjust the twiddles. This twiddle_offset parameter is defined in t_rtwo_fft in Table 2. It avoids the need for the complex multipliers section in Figure 8 and so the WFFT of Figure 10 results in the block diagram of Figure 11.

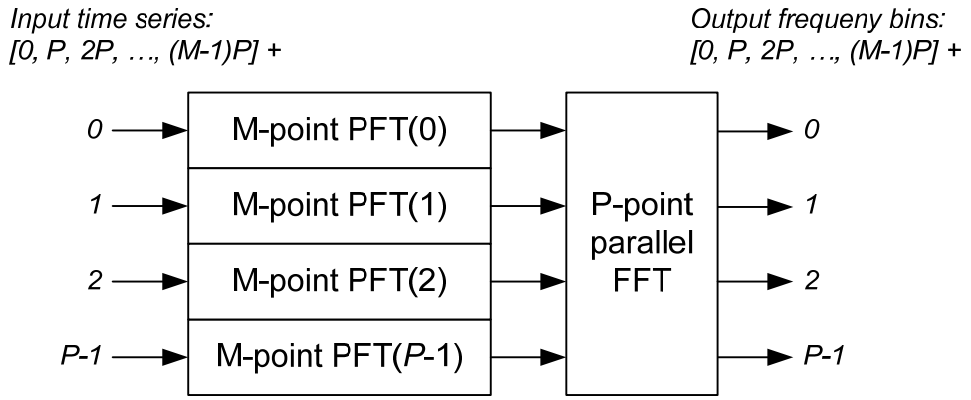

Figure 11: Block diagram for an N=MP-point wideband FFT

The parallel P-point FFT sections can be constructed by simply instantiating the appropriate number of butterfly sections from rTwoSDF in parallel with fixed twiddle factors per instance.

2.5.2 Parallel FFT entity and interface

The parallel FFT entity is called rtwo_fft.vhd. The parallel FFT must support the same functionality as the PFT, so N-points, arbitrary control of in_val, and able to operate for RTL latencies ≥ 0. Table 3 defines the interface signals for the rtwo_fft.vhd.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path reset
in_re_arr[0:*N*-1]	t_rtwo_slv_arr	IN	in_dat_w	Real inputs for index *t* is [0, 1, 2, …, *N*-1]
in_im_arr[0:*N*-1]	t_rtwo_slv_arr	IN	in_dat_w	Imaginary inputs for index *t* is [0, 1, 2, …, *N*-1]
in_valid	SL	IN	-	Input data valid strobe, same for all in array
out_re_arr[0:*N*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Real output X_{re}, index *f* is [0, 1, 2,…, N-1] or Separate output A_{re},B_{re} for two real inputs, index *f* is: [0,0, 1,1, 2,2, …, (N-1)/2,(N-1)/2]
out_im_arr[0:*N*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Imaginary output X_{im}, index *f* is [0, 1, 2,…, N-1] or Separate output A_{im},B_{im} for two real inputs, index *f* is: [0,0, 1,1, 2,2, …, (N-1)/2,(N-1)/2]
out_valid	SL	OUT	-	Output data valid strobe, same for all in array

Table 3: Interface signals for rtwo_fft (parallel FFT)

The rtwo_wfft.vhd should use the same single generic t_rtwo_fft as defined in Table 2, and ignore the wb_factor and twiddle_offset fields. The rtwo_fft.vhd must also support use_reorder and use_separate fields. Note that for the parallel FFT the ‘reorder’ operation becomes a rewiring function with t_rtwo_slv_arr in and out.

2.5.3 The WFFT entity and interface

The Wideband FFT entity is called rtwo_wfft.vhd. The wideband factor *P* > 1 implies that *P* data samples arrive in parallel per clock cycle. Table 4 defines the interface signals for the rtwo_wfft.vhd.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path reset
in_re_arr[0] in_re_arr[1] … in_re_arr[*P*-1]	t_rtwo_slv_arr	IN	in_dat_w	Real input, index *t* is *p* + [0, P, 2P, …, (M-1)P]: x_{re} or *a* for *p* = 0 x_{re} or *a* for *p* = 1 … x_{re} or *a* for *p* = *P*-1
in_im_arr[0] in_im_arr[1] … in_im_arr[*P*-1]	t_rtwo_slv_arr	IN	in_dat_w	Imaginary input, index *t* is *p* + [0, P, 2P, …, (M-1)P]: x_{im} or *b* for *p* = 0 x_{im} or *b* for *p* = 1 … x_{im} or *b* for *p* = *P*-1
in_valid	SL	IN	-	Input data valid strobe, same for all in array
out_re_arr[0] out_re_arr[1] … out_re_arr[*P*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Real output, index *f* is *p* + [0, P, 2P,…, (M-1)P]: X_{re} for *p* = 0 X_{re} for *p* = 1 … X_{re} for *p* = *P*-1 Separate output for two real inputs, index *f* is [*p*,*p*] + [0,0, P,P, 2P,2P, …, (M-1)P/2,(M-1)P/2]: A_{re},B_{re} for [*p*,*p*] = [0,0] A_{re},B_{re} for [*p*,*p*] = [1,1] … A_{re},B_{re} for [*p*,*p*] = [P-1,P-1]
out_im_arr[0] out_im_arr[1] … out_im_arr[*P*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Imaginary output, index *f* is *p* + [0, P, 2P,…, (M-1)P]: X_{im} for *p* = 0 X_{im} for *p* = 1 … X_{im} for *p* = *P*-1 Separate output for two real inputs, index *f* is [*p*,*p*] + [0,0, P,P, 2P,2P, …, (M-1)P/2,(M-1)P/2]: A_{im},B_{im} for [*p*,*p*] = [0,0] A_{im},B_{im} for [*p*,*p*] = [1,1] … A_{im},B_{im} for [*p*,*p*] = [P-1,P-1]
out_valid	SL	OUT	-	Output data valid strobe, same for all in array

Table 4: Interface signals for rtwo_wfft (wideband FFT)

The rtwo_wfft.vhd should use the same single generic t_rtwo_fft as defined in Table 2, but now with default *P* = 4 for the wb_factor field.

2.6 Wideband complex FFT for two real inputs

The WFFT for two real inputs reuses the ‘reorder’ and ‘separate’ functions from section 2.3.

2.7 PFS

The pre-filter structure (PFS) from LOFAR can be re-used. The pfs.vhd from LOFAR was ported to $UNB and is available at [13]. The LOFAR PFS implements *P*=1 and is described in [6]. Figure 12 shows the block diagram of the LOFAR PFS. Figure 13 shows the details of the FIR filter implemented by pfs_filter.vhd that is used in the LOFAR PFS.

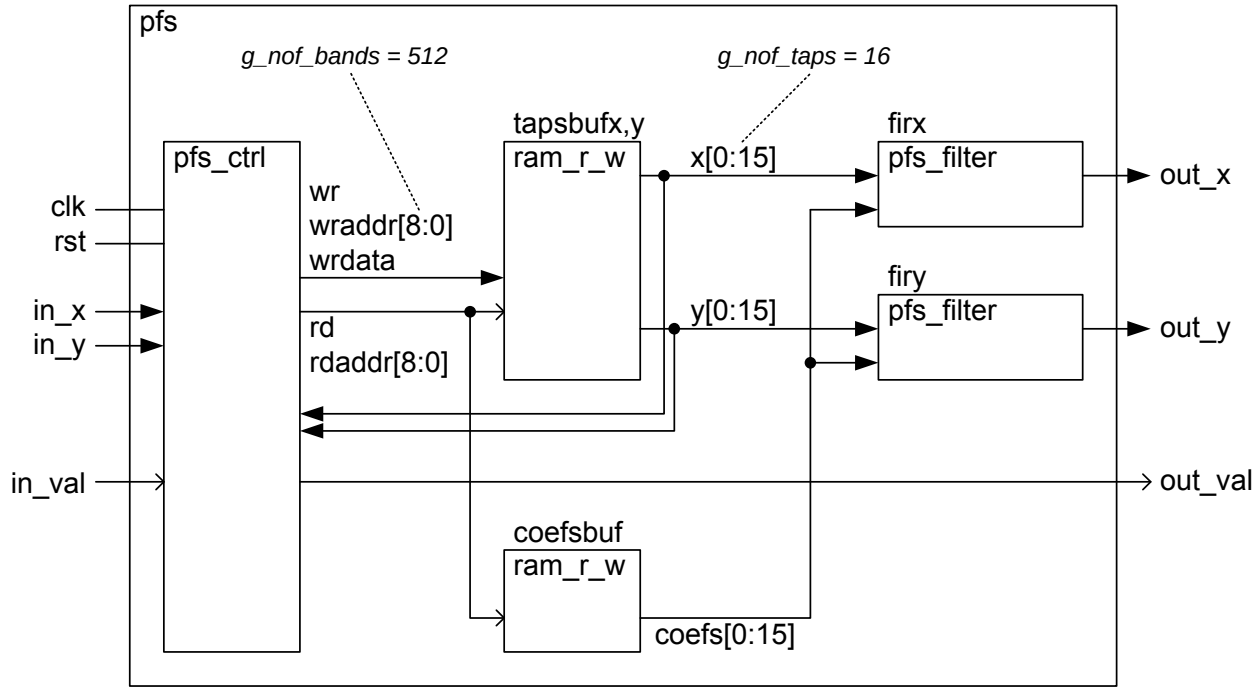

Figure 12: Block diagram of the LOFAR PFS implemented in pfs.vhd

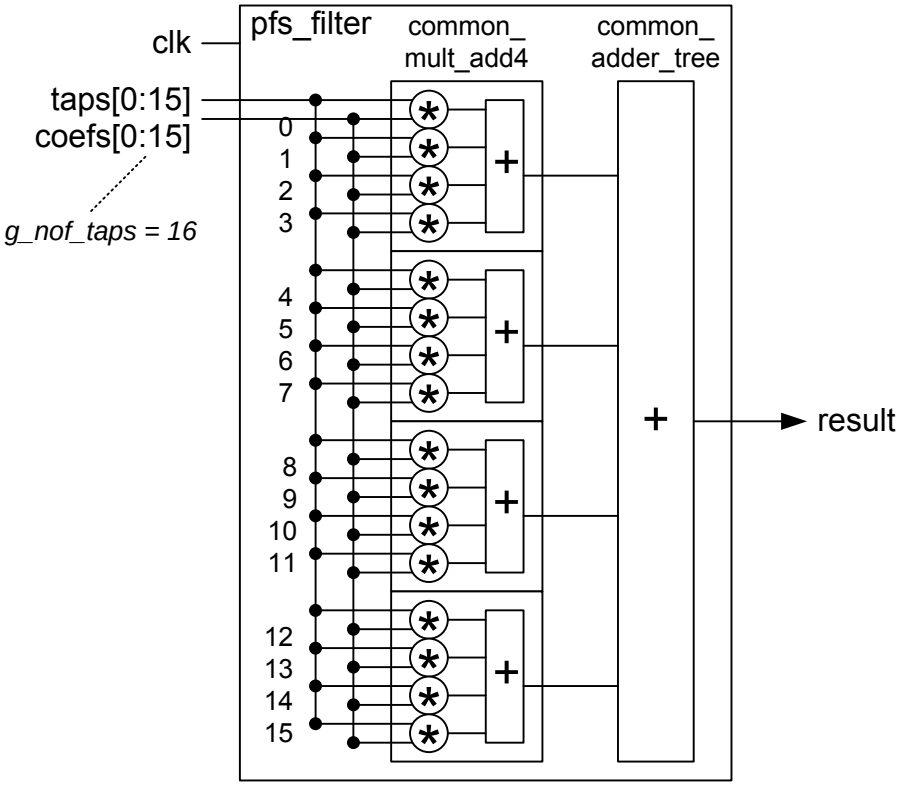

Figure 13: Block diagram of pfs_filter.vhd

2.7.1 PFS entity and interface

Table 5 and Table 6 show the interface signals and generic parameters of the LOFAR pfs.vhd.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path reset
dp_in.x	SLV	IN	g_dp_in_w	Input *x*
dp_in.y	SLV	IN	g_dp_in_w	Input *y*
dp_in.val	SL	IN	-	Input data valid strobe
dp_out.x	SLV	OUT	g_dp_out_w	Output *x* input *a* for *N*-point complex PFT with two real inputs
dp_out.y	SLV	OUT	g_dp_out_w	Output *y* input *b* for *N*-point complex PFT with two real inputs
dp_out.val	SL	OUT	-	Output data valid strobe

Table 5: Interface signals for LOFAR pfs.vhd

Generic	Type	Default	Description
g_nof_bands	Natural	512	Number of subbands is *N*/2
g_nof_taps	Natural	16	Number of FIR taps per subband
g_dp_in_w	Natural	8	Input data width
g_dp_out_w	Natural	12	Output data width (bit growth is ceil_log2(g_nof_taps)
g_coef_dat_w	Natural	16	FIR coefficients width
g_coef_init_file	String	-	Path to and name of coefficients RAM initialization file

Table 6: Generics for LOFAR pfs.vhd

2.8 Wideband PFS

The Wideband PFS may be constructed as sketched in Figure 14 using the pfs.vhd as building block. The pfs[1], pfs[2], and pfs[3] sections could use the FIR coefficients stored in the pfs[0] section to save RAM, because all *P* = 4 sections need the same FIR coefficients at the same time.

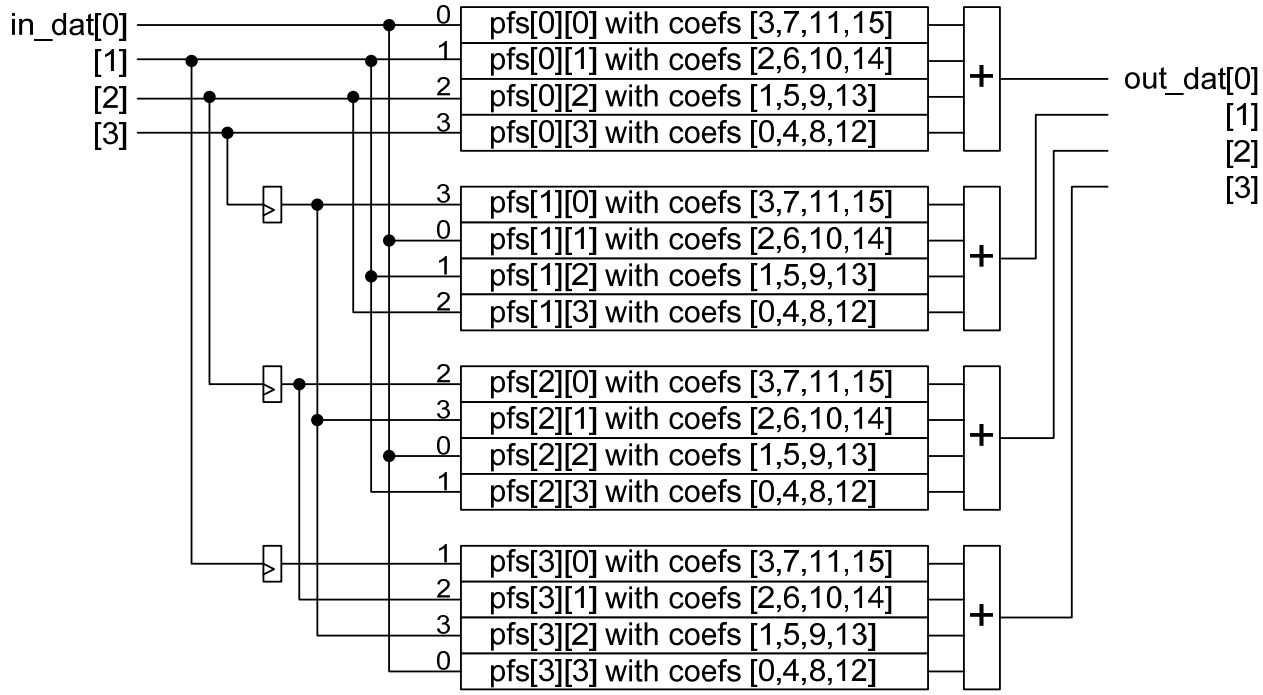

Figure 14: Sketch of the block diagram of the Wideband PFS for 16 taps and *P*=4

2.8.1 WPFS entity and interface

The WPFS entity is called rtwo_wpfs.vhd. Table 7 defines the interface signals for the rtwo_wpfs.vhd.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path reset
in_dat_arr[0] in_dat_arr[1] … in_dat_arr[*P*-1]	t_rtwo_slv_arr	IN	in_dat_w	Real input, index *t* is *p* + [0, P, 2P, …, (M-1)P]: *x* for *p* = 0 *x* for *p* = 1 … *x* for *p* = *P*-1
in_valid	SL	IN	-	Input data valid strobe, same for all in array
out_dat_arr[0] out_dat_arr[1] … out_dat_arr[*P*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Real output, index *t* is *p* + [0, P, 2P, …, (M-1)P]: *y* for *p* = 0 *y* for *p* = 1 … *y* for *p* = *P*-1
out_valid	SL	OUT	-	Output data valid strobe, same for all in array

Table 7: Interface signals for rtwo_wpfs

Generic	Type	Default	Description
wb_factor	Natural	1	Wideband factor *P*
nof_bands	Natural	512	Number of subbands is *N*/2
nof_taps	Natural	16	Number of FIR taps per subband
in_dat_w	Natural	8	Input data width
out_dat_w	Natural	12	Output data width
coef_dat_w	Natural	16	FIR coefficients width
coef_init_file	String	-	Path to and name of coefficients RAM initialization file

Table 8: Generic record type t_rtwo_pfs for WPFS

2.9 PFB for two real inputs

The PFB for two real inputs combines the LOFAR PFS from section 2.7 with the complex PFT for two real input from section 2.4 as shown in Figure 15 and in more detail in figure 1 in [1].

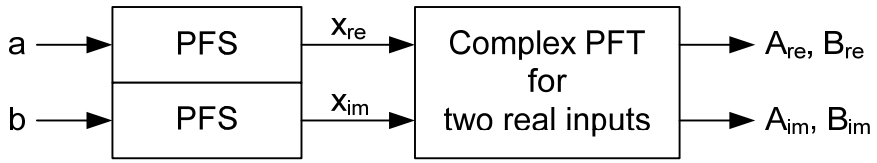

Figure 15: Block diagram of the PFB

Note that each subband has 1 complex sample per FFT slice, hence the downsample factor per subband is *N*, so the downsampling occurs at the Nyquist rate. This is called a critically sampled PFB and is the default for using an FFT with no input block overlap.

It may be useful to try this configuration in a test bench to prepare for the WPFB. However the WPFB also contains the PFB, because the WPFB must also suit *P*=1.

2.10 Wideband PFB for two real inputs

The WPFB has the similar structure as the PFB in Figure 15 but now using the Wideband variants for $P \geq 1$ of the WPFS and the WFFT.

2.10.1 WPFB entity and interface

The WPFB entity is called rtwo_wpfb.vhd. Table 9 defines the interface signals for the rtwo_wpfb.vhd. The rtwo_wpfb uses the generic records from t_rtwo_fft from Table 2 and t_rtwo_pfs from Table 8.

Signal	Type	I/O	Width	Description
clk	SL	IN	-	Data path clock
rst	SL	IN	-	Data path reset
in_dat_a_arr[0] in_dat_a_arr[1] … in_dat_a_arr[*P*-1]	t_rtwo_slv_arr	IN	in_dat_w	Real input *a*, see Table 7
in_dat_b_arr[0] in_dat_b_arr[1] … in_dat_b_arr[*P*-1]	t_rtwo_slv_arr	IN	in_dat_w	Real input *b*, see Table 7
in_valid	SL	IN	-	Input data valid strobe, same for all
out_re_arr[0] out_re_arr[1] … out_re_arr[*P*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Real output A_{re},B_{re}, see Table 4
out_im_arr[0] out_im_arr[1] … out_im_arr[*P*-1]	t_rtwo_slv_arr	OUT	out_dat_w	Imaginary output A_{im},B_{im}, see Table 4
out_valid	SL	OUT	-	Output data valid strobe, same for all

Table 9: Interface signals for rtwo_wpfb

3 Implementation specification

3.1 Firmware

Requirements:

1. The WPFB source code can be kept in dsp/rTwoSDF and all new VHDL files have prefix ‘rtwo_’ and use lower case names. Table 10 provides a preliminary list of the expected files.
2. The FPGA specific resources like multipliers and RAM must be instantiated via the VHDL wrapper components in common_lib.
3. The WPFB should be synthesized for the Stratix IV PFGA (type EP4SGX230KF40C2) using the Quartus II synthesis tool.
4. The WPFB should be able to clock at >250 MHz. The timing bottleneck must be clear and not trivial to improve.

File	Description
rTwoSDF.vhd	The already existing Radix-2 PFT component as described in [3, 15]
rtwo_separate.vhd	Separate function for two real inputs (section 2.4.2)
rtwo_pft.vhd	PFT is the pipelined FFT (section 2.4)
rtwo_fft	Parallel FFT
rtwo_wfft.vhd	Wideband FFT (section 2.5)
rtwo_wpfs.vhd	Wideband PFS (section 2.8)
rtwo_wpfb.vhd	Wideband PFB (section 2.10)
tb_rTwoSDF.vhd	Test bench for rTwoSDF
tb_rtwo_pft.vhd	Test bench for rtwo_pft (see tb_rTwoSDF and LOFAR tb_pft2.vhd)
Tb_rtwo_fft.vhd	Test bench for rtwo_fft (same stimuli and results as with tb_rtwo_pft)
tb_rtwo_wfft.vhd	Test bench for rtwo_wfft (same stimuli and results as with tb_rtwo_pft)
tb_rtwo_wpfs.vhd	Test bench for rtwo_wpfs (same stimuli and results as with LOFAR tb_pfs.vhd)
tb_rtwo_wpfb.vhd	Test bench for rtwo_wpfb.vhd (see LOFAR tb_pfb2.vhd)
r2sdf.qip	Quartus IP file in synth/quartus with all rTwoSDF source files
rtwo.qip	Quartus IP file in synth/quartus including r2sdf.qip and all other rtwo_* source files
rtwo_top.vhd	Wrapper entity in synth/quartus_top/ to obtain and maintain the synthesis results for the different components in the WPFB, similar as the LOFAR pf_top.vhd.

Table 10: Preliminary list of the expected WPFB VHDL source and test bench files

3.2 Verification requirements

Requirements:

1. The Wideband PFS must be verified by checking that an impulse input signal yields the FIR filter coefficients.
2. The Wideband FFT output must be compared to the (ideal) floating point FFT for a full scale, uniform noise input to show that the implementation loss (due to quantization and internal rounding) is conform what can be expected.
3. The Wideband PFB must be verified using a sinus signal or a stream of impulses with a period that does not integer divide the FFT size
4. All VHDL test benches must self check that the DUT VHDL works OK. If necessary use a golden reference result file for verification.
5. All VHDL test benches must support using 'run –all' using some tb_end signal to stop the clock.
6. The resource usage of the WPFB must be reported in a table (see for example pf_top.vhd in the LOFAR pft2 module).

Table 10 provides a list of existing testbenches for the rTwoSDF and for the LOFAR PPB2 that serve as examples for verifying the WPFB.

File	Description
tb_rTwoSDF.vhd	Testbench for rTwoSDF. The testbench can simulate: a) complex uniform noise input from a file generated by MATLAB testFFT_input.m b) impulse input from a manually created file Stimuli b) are useful for visual interpretation of the FFT output, because an impulse at the real input and zero at the imaginary input will result in DC and zero if the pulse occurs at the first sample or in sinus and cosinus wave if the impulse occurs at a later sample. However because the imaginary input is zero this does not cover all internals of the PFT implementation. Therefore stimuli a) are needed to fully verify the PFT. The rTwoSDF output can be verified in two ways: 1) The MATLAB testFFT_output.m can calculate the floating point FFT and compare it with the rTwoSDF implementation output file result. The testFFT_output.m als calculates the SNR value. 2) The rTwoSDF implementation output file is also kept in SVN as golden reference result to allow verification using a file diff command like e.g. WinMerge. This then avoids the need to run MATLAB to verify. The testbench asserts an error when the output does not match the expected output that is read from the golden reference file. The output is also written to a default output file to support offline analysis.
tb_pfs.vhd	Testbench for the LOFAR pfs.vhd. It verifies the impulse response of the PFS.
tb_pft.vhd	Testbench for the LFOAR pft.vhd. It can verify the output of the PFT for bit-reversed, normal complex or separated outputs. The input real and imaginary signal are selected by uncommenting the signal file name. The output gets automatically verified by checking that the output sample values do not differ too much from the golden reference values that were calculated using MATLAB. The input signal files and the golden reference result files are stored in the tb/data/ directory. The fft_responses_sketch.pdf [5] provides a sketch of the input signals and the expected FFT output signals.
tb_pfb2.vhd	Test bench for the polyphase filterbank PFS+PFT2 as used in LOFAR. After the PFB2 impulse response time of 16 FFT slices the 16*1024 taps PFS has flushed undefined internal status, so after that the PFB output subbands reflect the input. The test bench is self checking for: . a = "cosin_1.sig" has frequency 1 * fs/N so will appear in bin 1 . b = "cosin_39.sig" has frequency 39 * fs/N so will appear in bin 39
top.m	The top.m at Lofar/pft2/src/matlab provides a MATLAB simulation of the PFB2. It shows the response of the PFB2 for a sinus input.

Table 11: List of existing test benches

3.2.1 Verification approach for WPFS

A FIR filter like the PFS can be verified by checking that the impulse response yields the FIR coefficients [13]. This applies to both the PFS and the WPFS.

3.2.2 Verification approach for WFFT

For an FFT some fixed noise signal with golden result output verifies all internal states of the FFT as in tb_rTwoSDF.vhd. The noise signal FFT can also be calculated in floating point e.g. using MATLAB to determine the algorithm output as with testFFT_output.m [3, 15]. The difference between the implementation golden result output and the floating-point algorithm output must be sufficiently small [1, 14] and is called the implementation loss. To verify the scaling in the FFT (i.e. no overflow) it is also necessary to use an impulse at t_0 and a sinus wave [14]. Using a shifted impulse response e.g. at t_1 is also interesting, because that results in a sinus output. The stimuli and expected results for the WFFT are the same as for the PFT, but in a different data format.

3.2.3 Verification approach for WPFB

The WPFB can be verified as in the LOFAR tb_pfb2.vhd selecting a waveform from a file. However it seems more appropriate to verify the WPFB using a sinus generated with the diag_wideband_wg. Two cases are interesting. First use a sinus that has a period that divides the FFT size *N*. Second use a sinus with a frequency that falls somewhere between two frequency bins. The WPFB has an impulse response of nof_taps * *N* samples, so after that the output will become stable.

3.3 Validation requirements

There are no validation requirements other then that the WPFB component must synthesize OK. Hence no reference design needs to be made to validate the WPFB on hardware.

4 Deliverables

4.1 Firmware

All source code must be available in the UniBoard SVN repository [12]. The code must compile, simulate and synthesize correctly.

4.1.1 Not to do (yet)

The following features are not required for the Apertif WPFB, so therefore they are not required and should not be done yet.

- Saving RAM for the reorder function (see section 2.4.1).
- Improving the multiplier usage by using a biplex FFT (see section 2.4.4).
- Investigating rounding instead of truncation in rTwoSDF (see section 2.3.3).
- Scramble the rounding crosstalk that occurs between two real inputs using the (un)switch feature from the LOFAR PFT [6]
- The PFS FIR coefficients for the two real inputs of the PFB are the same, so they could be shared to save memory.
- PFS with multiple ports that can potentially share the FIR coefficients to save memory
- FFT with multiple ports that can potentially share the twiddle factors to save memory
- Support WPFS and WPFB for wideband factors *P* that are not a power of 2.
- An oversampled or overlap PFB using a downsample factor that is less than the FFT size

4.2 Documentation

The APERTIF WPFB must be documented according to the ASTRON report module document template.