A 16 channel FFT multiplexer

G. Comoretto[1], A. Russo[1], G. Tuccari[2]

[1]INAF - Osservatorio Astrofisico di Arcetri
[2]INAF - Istituto di Radioastronomia, sez. di Noto

Arcetri Technical Report N° 1/2009

Abstract

A common problem in radio applications is the need to divide a larger bandwidth into smaller, contiguous frequency channels, in order to analyze, transmit, or store the signal using slower equipments.

Here a frequency multiplexer based on a polyphase filter and FFT structure is described. The input signal has a 512 MHz bandwidth, sampled by a fast 1.024 GS/s ADC, and the output signals are 15 parallel VLBI data streams, with a bandwidth of 32 MHz each. Both input and output signals are real.

The instrument is implemented as a single CORE2 board on the DBBC VLBI digital data acquisition terminal.

1 Problem definition

A common problem in signal processing is the so called *multiplexing in frequency*, in which a wideband signal is divided into possibly contiguous sub-bands (here *channels*), each one representing a portion of the input bandwidth.

In this way it is possible to resample each channel at a fraction of the input sampling frequency, so that it could be recorded, analyzed, etc. by slower, and simpler, electronics. As electronics complexity usually increases quadratically with sampling frequency, it is often convenient to treat the signal with N parallel components, one for each frequency channel, than with a single, but N times faster, component, or with N parallel components each one analyzing a fraction of the input samples (multiplexing in time).

For example, a multiplexed in time spectrometer with a given resolution of 1000 spectral points can be built using N identical spectrometers, each with 1000 spectral points and fed with a contiguos segment of samples (e.g. 1 ms each). After N periods, a spectrum of the whole dataset is reconstructed by averaging the N individual spectra. Using a division in frequency approach, each spectrometer analyzes only a subset 1/N of the input frequency range, and needs just 1000/N spectral points.

A simple way to perform the multiplexing in frequency operation is thus quite useful. The usual approach is to use N individual heterodyne receivers, each one tuned to extract a given portion of the input band. In recent years, approaches based on the Fourier transform algorithm are becoming popular, due to their intrinsic simplicity. The so called "polyphase filter" concept, that allows to shape the band of the spectral channels in a controlled way, and the use of fast digital devices, is used to overcome the problem of poor band edges and limited channel-to-channel insulation of the standard Fourier transform.

This work describes the implementation of a FFT based $N = 16$ band splitter to be implemented in the *digital Baseband Converter* (DBBC) hardware developed by the EVN. The DBBC is a modular system composed of 1 to 4 fast ADCs, with an input bandwidht of 0.5 to 1 GHz each, that interfaces with the MK5a VLBI recorder. Data processing is performed by up to 16 FPGA-based boards. The proposed implementation can be used for VLBI recording of a wideband signal, up to 470 MHz, using up to 15 VLBI data streams each 32 MHz wide (nominal). The implementation is very compact, using just a single CORE2 processing board.

The problem is described mathematically in chapter 2, and the actual implementation is described in chapter 3.

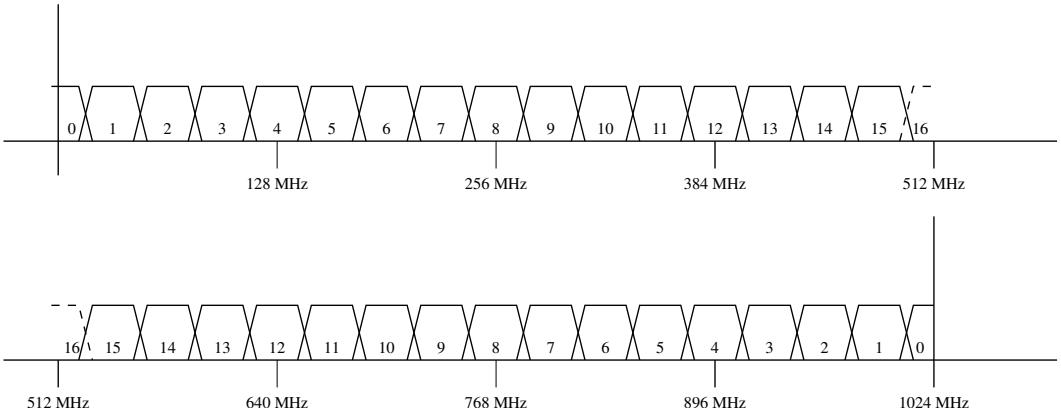

Figure 1: Conceptual description of the signal processing in the polyphase filterbank for an input signal in the 1^{st} Nyquist range (0 to 512 MHz) and in the 2^{nd} Nyquist range (512 to 1024 MHz)

In fig. 1 the signal processing is shown in the frequency domain. The input signal, either in the frequency range 0-512 MHz (upper graph), or in the 512-1024 MHz range (lower graph) is splitted into 17 spectral bands. Band 16 is discarded, and band 0, with half bandwidth, is just filtered. Channels are placed side-by-side, with a small unusable portion between them.

2 Mathematical formulation

The FFT frequency divider must provide 16 (15 usable) real outputs at 1/16 sample rate from a time multiplexed real input.

The real input format is a 8x data stream, in which 8 consecutive samples are presented at each clock cycle to the system. The system clock is thus 1/8 of the sampler clock, and 2 times the output data rate. As an output sample is produced every two clock cycles, it is possible to perform the computation separately on odd and even samples.

The mathematical processing will be first analyzed in the most direct form. For each output channel $k, k = 0 \ldots 15$, the signal will be:

- multiplied by a complex exponential, $\exp(-2\pi ijk/32)$, where j is the time index of the input data stream, in units of the sampler clock f_c
- filtered by a finite impulsive response (FIR) filter, with a low pass response and a cut-off frequency of $f_c/64$
- decimated by a factor of 16. The resulting data rate is $f_o = f_c/16$, and the filtered data occupies the frequency interval $[-f_o/4, f_o/4]$
- upconverted by $f_o/4$, by multiplying the complex data stream by $\exp(2\pi il/4)$, with l the time index of the output data stream, in units of the output clock f_o. The frequency interval extends from 0 to $f_o/2$.
- converted to real, discarding the imaginary part. The output data stream X_k is correctly sampled, without aliasing, as there are no frequency components above $f_o/2$ and at negative frequencies.

The net result is that each output data stream, k corresponds to a frequency slot in the range $(k - 1/2)f_c/32$ to $(k + 1/2)f_c/32$, converted to the upper sideband. The stream with $k = 0$ represents the portion of the input spectrum with $f < f_c/32$, and a equivalent portion near $f_c/2$ is not present in any stream.

The direct computation described above is very inefficient. Most of the intermediate results are discarded, or computed multiple times. Performing the first multiplication before the filtering forces the latter to be performed on complex, instead of real, values. The frequency conversion for all output channels can be performed more efficiently using a Fast Fourier Transform algorithm. Therefore the implementation has been completely modified, at the point that the above algorithm is barely recognizable.

The overall computation must always correspond, however, to the formula

$$X_k(l) = \mathrm{Re}\left[\exp\left(\frac{\pi}{2}il\right)\sum_p x(Nl-p)t(p)\exp\left(-2\pi i\frac{(Nl-p)k}{2N}\right)\right] \tag{1}$$

where N is the number of output channels, $x(i)$ are the input samples, $z_k(l)$ are the output samples (decimated) for channel k, and $t(p)$ is the impulse response for the filter.

2.1 Polyphase filtering

If $x(j)$ is the input data stream, and assuming that we want to divide it into N independent frequency slots, after the frequency conversion we obtain for the data stream $k, k = 0 \ldots (N-1)$ the signal

$$x'_k(j) = x(j)\exp\left(2\pi i\frac{-jk}{2N}\right) \tag{2}$$

After filtering $x'(j)$ using a filter with impulsive response $t(p)$ (in the time domain), we obtain

$$x_k"(j) = \sum_p x'_k(j-p)t(p) = \sum_p x(j-p)\exp\left(-2\pi i\frac{jk}{2N}\right)\exp\left(2\pi i\frac{pk}{2N}\right) \tag{3}$$

If the filtered signal is computed only a time $j = Nl$, i.e. is decimated by a factor N, and the index p for the tap is decomposed as $p = q + 2Nr$, with $q = 0, \ldots 2N$, the above formula can be rewritten as

$$x_k"(l) = (-1)^{lk} \sum_{q=0}^{2N-1} y_q(l) \exp\left(2\pi i \frac{qk}{2N}\right) \quad (4)$$

$$y_q(l) = \sum_r x\left(Nl - q - 2Nr\right) t(q + 2Nr) \quad (5)$$

The first operation (apart for the alternating sign) is a Fourier transform of length $2N$, and the second is a series of $2N$ short filters, each one with tap coefficients that are a subset of the original ones. The alternate signs in the Fourier transform produce a frequency reversal in the output channels with k odd, but this can be easily corrected in the conversion to real stage.

2.2 Fourier transform

The Fourier transform can be further optimized by considering that the clock frequency is twice the desired output sample frequency, and that the input signal is real.

2.2.1 Decimation in time FFT

In a division-in-time architecture, odd and even samples are processed separately by two half length transforms, and then combined together.

$$(-1)^{lk} x_k"(l) = \sum_{q=0}^{N-1} y_{2q}(l) \exp\left(2\pi i \frac{qk}{N}\right) + \sum_{q=0}^{N-1} y_{2q+1}(l) \exp\left(2\pi i \frac{(q+1/2)k}{N}\right) \quad (6)$$

$$= z_k^e(l) + \exp\left(2\pi i \frac{k}{2N}\right) z_k^o(l) \quad (7)$$

$$z_k^e(l) = \sum_{q=0}^{N-1} y_{2q}(l) \exp\left(2\pi i \frac{qk}{N}\right) \quad (8)$$

$$z_k^o(l) = \sum_{q=0}^{N-1} y_{2q+1}(l) \exp\left(2\pi i \frac{qk}{N}\right) \quad (9)$$

Only values of $k \leq N$ must be computed, so only one leg of the "butterfly computation" (that with the + sign) needs to be computed, as described in the above equation.

As y_q is real, the quantities z_k^e and z_k^o are hermitian, i.e. $z_{N-k} = z_k^*$, where the asterisk denotes the complex conjugate. z_0 and z_N are real. The FFT processor takes N real inputs, and delivers $N - 1$ complex plus 2 real outputs.

2.2.2 Winograd Fourier Transform algorithms

The Winograd short-length FFT is an algorithm that decomposes the Fourier Transform into three matrices, represented as:

$$\overrightarrow{X} = T_N \cdot \overrightarrow{x} = B_N D_N A_N \cdot \overrightarrow{x} \quad (10)$$

where B_N and A_N are incidence matrices containing only numbers 1, 0 and -1: multiplications by these matrices can be computed with only additions and subtractions. D_N is a diagonal matrix and thus requires at most N multiplications. These features minimize the multiplicative complexity. Practical algorithms have been written for several short lengths: $2, 3, 4, 5, 7, 8, 9$ and 16. Table 1 summarizes the number of multiplications and additions used to compute DFT for these small N, including the ones for $W_N^0 = 1$, with $W_N^k = \exp\left(\frac{2\pi i k}{N}\right)$ and $k = 0 \ldots (N-1)$ [3][5].

Larger numbers algorithms can be obtained for N having more than one prime divisor. In those cases the computation of DFT of $N = N_1 + N_2$ points can be decomposed into computing the DFT for N_1

N	multiplications	multiplications by W_N^0	additions
2	0	2	2
3	2	1	6
4	0	4	8
5	5	1	17
7	8	1	36
8	2	6	26
9	12	1	44
16	10	8	74

Table 1: DFT for small N

points in which each multiplication is replaced by computing the DFT of N_2 points. But for large N a direct application of the Winograd FFT algorithm entails a prohibitively large number of additions. However hybrid strategies can be adopted using small-size Winograd algorithms and FFT algorithm based stages.

2.3 Conversion to real

The output real signal is given by

$$X_k(l) = \mathrm{Re}\left[\exp\left(\frac{\pi}{2}il\right) x_k"(l)\right] \tag{11}$$

The FFT processor produces, every two cycles, the two complex values z_k^e and z_k^o. For each k, from these two values two real samples X_k and X_{N-k} can be computed, using relations 7, 11 and the fact that $z_{N-k} = z_k^*$:

$$X_k(l) = (-1)^{lk}\mathrm{Re}\left(\exp\left(\frac{\pi}{2}il\right)\left[z_k^e(l) + \exp\left(2\pi i\frac{k}{2N}\right) z_k^o(l)\right]\right) \tag{12}$$

$$X_{N-k}(l) = (-1)^{lk}\mathrm{Re}\left(\exp\left(\frac{\pi}{2}il\right)\left[z_k^{e*}(l) - \exp\left(-2\pi i\frac{k}{2N}\right) z_k^{o*}(l)\right]\right) \tag{13}$$

In the second relation, it has been assumed that N is even, for simplicity.

The first exponential assumes only the values ± 1 and $\pm i$, and the sign can be absorbed in the first factor. The second exponential corresponds to a linear combination of the *twiddle factors*, $W_k^r = \cos(\pi ik/N)$ and $W_k^i = \sin(\pi ik/N)$ Using the suffixes r and i to denote the real and imaginary parts of the quantities y_k, one obtains, for l even:

$$X_k(l) = (-1)^{lk+a}\left(z_k^{er}(l) + W_k^r z_k^{or}(l) - W_k^i z_k^{oi}(l)\right) \tag{14}$$

$$X_{N-k}(l) = (-1)^{lk+a}\left(z_k^{er}(l) - W_k^r z_k^{or}(l) + W_k^i z_k^{oi}(l)\right) \tag{15}$$

The term a in the first exponent takes into account the sign of the first exponential: $a = 0$ for $l = 0, 1$ module 4, and $a = 1$ for $l = 2, 3$ module 4. The corresponding relations for l odd are:

$$X_k(l) = (-1)^{lk+a}\left(-z_k^{ei}(l) - W_k^i z_k^{or}(l) - W_k^r z_k^{oi}(l)\right) \tag{16}$$

$$X_{N-k}(l) = (-1)^{lk+a}\left(z_k^{ei}(l) - W_k^i z_k^{or}(l) - W_k^r z_k^{oi}(l)\right) \tag{17}$$

The cases with $k = 0$ and $k = N$ are peculiar, as the input signal is real and the bandwidth is half that of the other channels. Frequency translation is not required, and usually applied only to the $k = N$ channel. In most applications, these signals are just discarded, considering also that the extremes of the

input bandwidth are usually affected by other ill factors (e.g. rolloff and aliasing in the input analog filter).

For $k = 0$ the output signal is just the sum of the odd and even samples, $X_0(l) = z_0^o(l) + z_0^e(l)$. The signal is just the first $1/2N$ portion of the input band, low-pass filtered.

For $k = N/2$, y^e and y^o are real, and $X_N(l) = (-1)^a(z_n^{er}(l) - z_n^{or}(l)$. If the alternating sign is omitted, the frequency band is reversed, with output frequency zero corresponding to the higher sampler frequency. If it is present, the band is represented in natural order.

Both channels have a sample frequency that is half that of the remaining channels, as they span a frequency range up to frequency $1/2N$. Samples are produced during the "even" cycle, and last for two cycles of the output clock.

3 Implementation

The algorithm has been implemented on CORE1 and CORE2 boards, but due to resource limitations the final, fully operating design has been implemented only in the CORE2. These boards host a single large FPGA, respectively of the Xilinx Virtex2 and Virtex4 families.

The code has entirely been written in the VHDL programming language, avoiding any specific dependance on the Xilinx hardware. The design functionality has been extensively simulated using the Aldec FPGA tools, that provide also a programming environment and a common interface to all the other tools.

The code has been converted to a Xilinx netlist using the Synplify synthesis program, and then translated to a physical design in the target chip by proprietary tools. Although the Xilinx synthesis tool does not detect any formal error in the code, the synthesized code presents several errors, especially in the ROM tables used for FFT twiddle coefficients, and the resulting design is unusable. Use of a good synthesis tool is therefore mandatory.

The device has been designed as a black box, with generic input and output signals, independent from the details of the chip input/output structure. A *framework* structure, dealing with physical board details, clock distribution, signal interfacing, and computer programming has been developed separately[1], and merged with the design after simulation.

The high speed input/output bus, carrying ADC signals, require particular care to meet timing constrains. A specific constrain file has been used to force all components operating at 256 MHz to be placed in specific chip locations.

3.1 Polyphase filter

The low-pass filter response is probably the most important parameter of the instrument, as it determines passband flatness, transition region between channels (i.e. usable portion of the input band), and insulation among different channels. The available hardware resources impose constrains on the number of available tap coefficients, and on the resolution of their representations. FPGAs contain hardware multipliers, but their number is constrained, and multiplication by a fixed coefficient of typically a small size is better performed using a multiplier implemented with discrete logic.

For practical designs the number fo available taps is of the order of a few 100s, with tap representation using 8-12 bits.

The stopband is determined mainly by the number of bits in the tap coefficients representation. To accurately cancel off-band signals, the individual tap coefficients must be close to the design specification. Considering the typical performance of a VLBI terminal, an insulation of 40-45 dB has been considered sufficient. This can be achieved using 8 bit coefficients. Each additional bit can increase the stopband rejection by 6 dB, but the filter size must also be increased.

In table 2 three filters have been calculated. For each filter, the length, stopband and passband have been fixed, and the tap coefficients have been computed using the Remez algorithm. The passband ripple has been kept constant, while attempting to increase the stopband attenuation by giving a much higher weight to the stopband specification in the fitting algorithm.

The tap coefficients have then been truncated to a number of bits comprised between 8 and 12, and the solution with a degradation of less than 4 dB with respect to the *infinite resolution* has been chosen.

In figures 2 the response for the three filters has been plotted.

Length (taps)	Tap res. (bits)	Pass (f_c)	Stop (f_c)	Usable BW	Ripple (dB)	Min rej. (dB)	Typ rej (dB)
256	∞	0.012	0.019	77%	0.5	45	45
256	8	0.012	0.019	75%	0.8	42	45
384	∞	0.013	0.0183	83%	0.6	52	52
384	9	0.013	0.0183	85%	0.6	48	52
512	∞	0.0135	0.0178	86%	0.6	58	58
512	10	0.0135	0.0178	86%	0.06	54	58

Table 2: Parameters of the prototype low-pass filter, 8, 9 and 10 bit version

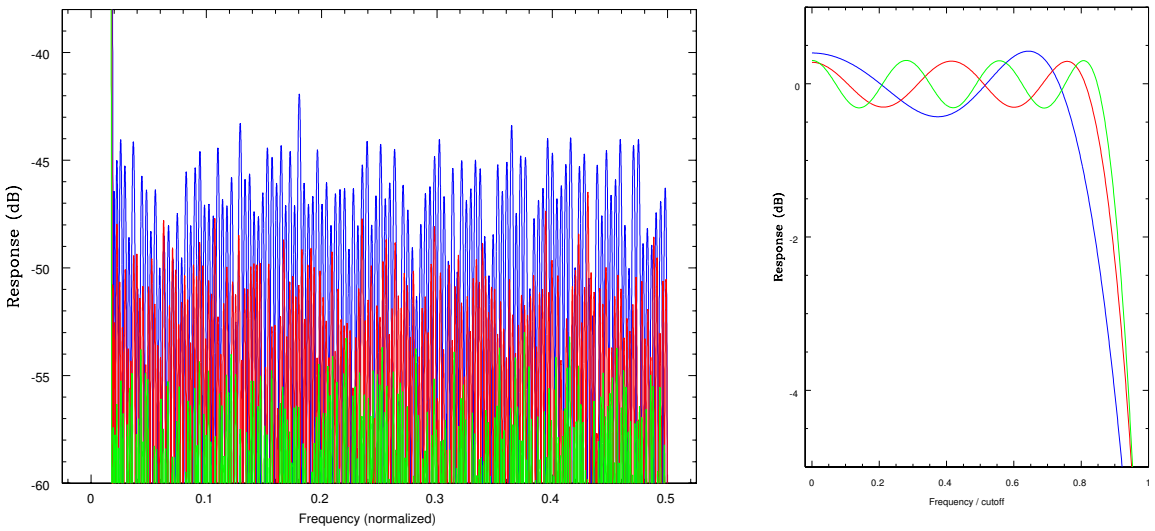

Figure 2: Stopband and passband response of prototype low-pass filter for 8 bit, 256 taps (blue), 9 bit, 384 taps (red) and 10 bit, 512 taps (green)

The final design adopts the filter with 512 taps, for a minimum stopband rejection of 54 dB.

The filter tap coefficients have been converted to a "package" VHDL file. The package provides a series of constant values, of which the most important are the number of taps and the number of bits in the representation. A function returns the tap coefficient value, given the tap index.

The VHDL file that implements the polyphase filter has been parameterized using these values, so different filters can be obtained just by changing the package file.

The top level file instantiates an array of 8 *legs*, one for each of th 8 time multiplexed samples in the input data stream. In each leg, the input sample feeds 4 FIR filters, one for input index p and the second for index $p + N$.

On odd and even cycles, tap coefficients are changed in order to compute both $y_o(p)$ and $y_e(p)$. These values are presented in turn to the FFT processor, that computes on alternate cycles the quantities z_o and z_e of eq. 7.

The maximum size of a tap product is equal to $7 + n_b$, with n_b the number of bits in the tap representation. In each of the 32 legs, however, only the central taps have the full size, and the total filter output has a size of at most $8 + n_b$. For white noise, the RMS amplitude of the signal is increased roughly by $n_b - 1$ bits. Even for the 10 bit filter, this means that if the signal can be represented with the 8 input bits, the filter output can be represented using about 17 bits. The input word size of the FFT processor, 18 bit, is never exceeded.

To prevent overflow in the FFT stage, however, it is advisable to keep at least 2 bits of growth margin.

In the filter implementations with $n_b > 8$, 1 or 2 bits are thus truncated. Truncation introduces a DC bias in the system, that affects the X_0 output. As this is usually discarded, no correction for the bias has been considered. *NOTE: this is not implemented in the current FFT design.*

3.2 FFT processor

The FFT processor is a standard design, with 16 inputs (real) and 16 outputs (complex). The internal design has been optimized to avoid unnecessary computation of null imaginary components, and of unused outputs.

Two designs have been considered: a standard division-in-time FFT algorithm, and a base-16 short length Winograd algorithm. Due to short development time, the simpler DIT FFT has been adopted. The Winograd algorithm may be added in a second time. It uses less multipliers, and less processing stages, thus reducing rounding errors. The spared multipliers can be used for other components in the chip, or for increased filter performance.

Both FFT designs include an overflow detection system. If an overflow occurs in any stage, the `ovf` signal is set to 1. The signal is latched and can be read using the control register. It is also routed to one of the output LEDs, in order to have a visual feedback during operations.

3.2.1 Division in time FFT

The division in time FFT is composed of four stages. The first stage has all real inputs, and since the twiddle coefficients are all ± 1 no multipliers are needed, and the result is still real. The second stage has complex outputs, but still no multipliers are needed (twiddle coefficients ± 1 and $\pm i$). These two stages are implemented with dedicated VHDL files.

The remaining two stages use a parametrized VHDL code, allowing for easy implementation of FFT blocks of any size.

Twiddle coefficients are computed using a dedicated VHDL package.

The last stage has only the first half of its outputs connected. No special code has been written to exploit this, as the synthesis routine automatically simplifies the design deleting the unused components.

3.2.2 Winograd algorithm

An alternative implementation of the FFT has been developed using the Winograd 16 points algorithm. The used matrices B_{16}, A_{16} and D_{16} are shown in figures 3 and 4 [4].

$$\mathbf{B}_{16} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & -1 & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Figure 3: B_{16} 16×18 matrix of 16 Winograd short length algorithm.

The resulting VHDL code is quite complex, not having the recursive structure of the FFT. The matrices A_N and B_N aren't squared, their sizes are respectively: $18X16$ and $16X18$. Consequently the input data is expanded slightly when multiplied by A_N matrix and contracted back to the original size when multiplied by theB_N matrix. The advantage of this method is on the total resource usage, that is considerably reduced.

$$\mathbf{D}_{16} = \mathrm{diag}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \cos 2u \\ \cos 2u \\ \cos 3u \\ \cos u + \cos 3u \\ \cos 3u - \cos u \\ -i \\ -i \\ -i \\ -i\sin 2u \\ -i\sin 2u \\ -i\sin 3u \\ -i(\sin u - \sin 3u) \\ -i(\sin u + \sin 3u) \end{bmatrix} \quad \mathbf{A}_{16} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Figure 4: D_{16} diagonal elements of 16 Winograd short length algorithm, where $u = 2\pi/N$ with $N = 16$, and A_{16} 18×16 matrix

3.3 Conversion to real

The output stage combines together the odd ad even FFT results, z_K, to produce the two output streams X_k. Examining equations 14-17, it is apparent that the result is the sum of either the real or imaginary part of y_k^e, and a linear combination of the real and imaginary part of y_k^o, weighted with the A simple architecture to implement this is described in figure 5.

The module requires as input the phase of the conversion exponential, i.e. the index l module 4, and the odd/even clock. The resulting signal is a value ranging cyclically from 9 to 7, incremented at each clock.

The even input, $y_k^e(l)$, is delayed by one clock, to put it in phase with the odd input, and the phase is used to select the real or imaginary part. and to change its sign as needed. Manipulating the sign, it is also possible to convert the frequency scale of the output signal from USB to LSB.

A 8x2 memory is used to select the coefficients for the real and imaginary part of the odd input, including the appropriate sign. The two multiplications are performed using 18 bit hard multipliers. The three products are then summed together, and stored in two output registers. The two output streams are out of phase by one clock, but can be re-phased in the following stages, by enabling subsequent operations only on even cycles.

3.4 Output stage

The X_k output signals are represented with 18 bit, much more than needed, and may have widely different amplitudes if the input band is not perfectly equalized. Therefore for each signal one must:

- Measure the total power integrated over the output band, and some time interval (typically 1 second)
- Quantize the signal with 1 or 2 bit representation, as required by the VLBI correlator, using threshold adjustment appropriate for the measured RMS amplitude

The output coding is defined in the MARK5 standard definition document. Each output sample is coded as a 2 bit quantity. Bit with lower index is the sign bit, and the one with higher index is the magnitude bit. Code representation is binary offset, as shown in table 3, together with the expected statistics in each code for a Gaussian noise.

Output samples are available on the HSO bus, sent over the output VSI connectors. Bits HSO(00–31) are used for the 16 output channels 0 to 15, with channel 16 not used. Channel 0 is sent to lines HSO(00) (sign) and HSO(01) (magnitude), and channel 15 to lines HSO(30) and HSO(31).

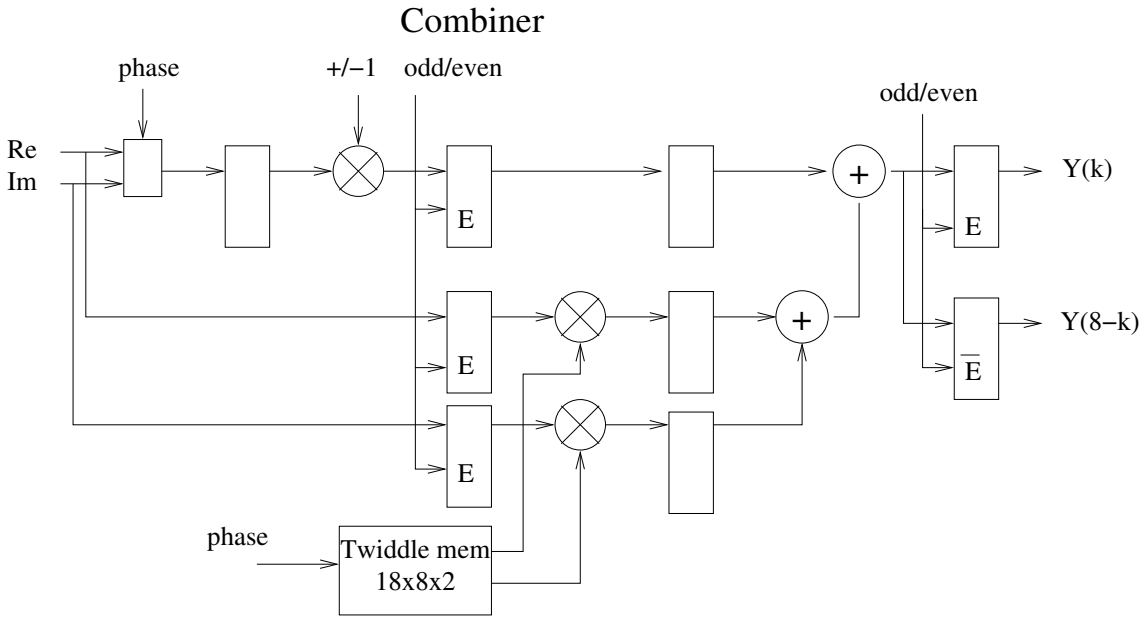

Figure 5: Structure of the conversion-to-real block

	-high	-low	+low	+high
sign/magnitude code	00	01	10	11
Statistics for optimal quantization (%)	18.2	31.8	31.8	18.2

Table 3: Coding and statistics for output samples

The VSI clock runs at fixed 64 MHz, with the rising edge exactly at the center of each sample. VSI1PPS is held high for one cycle every second (64 million samples). VSI valid bit (PVALID) is always set.

All signals for the second VSI bus are copied from the corresponding HSO input lines of the board.

In figure 3.4 the output of channel 9 (752 to 720 MHz) is shown for an input tone at 748 MHz. The traces show the monitor (DAC) output, and the sign, magnitude and clock signals on the VSI bus. The output signal period is 250 ns, as expected. The cursor is placed on a transition of the magnitude-sign bits, corresponding to the falling edge of the VSI clock.

3.5 Output LEDs

The board has 16 user-programmable LEDs. They have been assigned to the functions described in table 4

LED 01 is set when the register 63 of the board is addressed, and cleared when any other register is addressed. It resets all DLL's in the board. This signal is completely asyncronous, as the clock signal is not available during DLL reset, and must be explicitly cleared to allow the board to operate.

Led 02, 03 and 09–12 are used to monitor the status of the internal DLL's

Led 04 blinks at 1 Hz during normal operation, in sync with system 1PPS.

Led 05 can be used to quickly adjust input signal level, in order to avoid saturation in the FFT block.

Leds 06, 07 and 08 can be used to check the control interface. Led 06 should blink at each computer access, and 07-08 copy bits 20-19 of the PCI7200 output register (not used for the addressing scheme).

Leds 13–16 can be used to check the correct addressing of the board. At the moment the board responds to the fixed address 0001, but in future releases the address will be set using the rotary switch on the board.

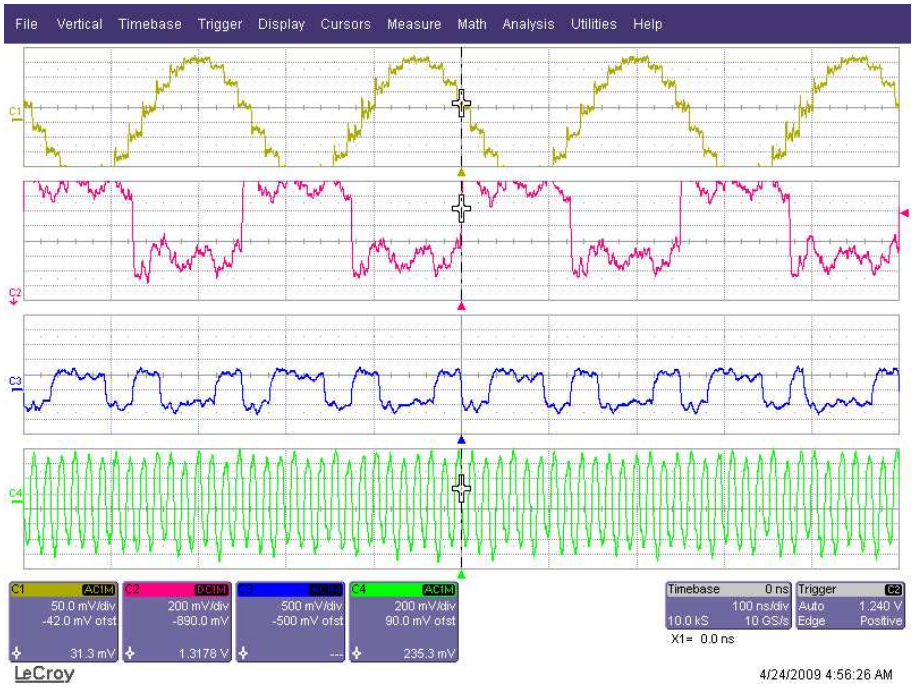

Figure 6: Example fo output converted signals, for an output tone at 4 MHz. From top: analog monitor, sign, magnitude, clock

4 Programming interface

The filterbank requires an interface with the control computer, in order to be able to read total power values, and to program the equalization scale factors. Other required functionalities include the ability to synchronize to an external PPS signal, and monitor of the overflow status.

The interface is composed of a series of programmable registers, on the FPGA chip, and a set of programming routines, written in C++ using an object oriented methodology.

4.1 Hardware description

In the adopted framework architecture, each chip contains up to 4 independent programming blocks, of 64 words each. The board is identified by a 5 bit address, so a complete address is composed of 4 parts, specified as the upper 16 bits of the PCI7200 output word `wd`:

LED	Fucntion
01	PLL Reset signal
02	LOCK status for DLL at 256 MHz
03	LOCK status for DLL at 128 MHz
04	1PPS: set for 0.1 second every internal 1PPS
05	Overflow: set if some overflow condition detected
06	Bus Activity: set for 0.1 s at each computer access
08-07	Bit 20-19 of PCI7200 output register
12-09	Status word for DLL at 128 MHz
16-13	Address selector (from rotary switch)

Table 4: Assignment for board LEDs

- Board address: bits 31–27 of `wd`
- Register address: bits 26–21 of `wd`
- Block address: bits 18-17 of `wd`
- Read enable bit: bit `wd(16)`

The read enable bit specifies that the operation is a *read only* operation, i.e. registers are not modified but only read. A read operation is always assumed at each computer access, i.e. the addressed register is always placed on the PCI7200 input bus.

The filterbank block has an address space of 32 read/write registers, at block address 0. Only registers 0, and 16–31 are actually implemented, the remaining do not physically exist. They are listed in tab. 5.

Register	Write value	Read value
0	Control register	Status
1-15	unused	
16-31	Threshold level	TP read

Table 5: Programming interface

All write registers are 16 bit in size, even if not all bits are used, and all read registers are 32 bit wide.

Register 0 is used to program the control register, and read the status register. Its bit definition is given in tab. 6.

Bit	Control register value
3-0	Monitor DAC select
4	Input select: 0=ADC, 1=line
5	1PPS Sync enable
15-6	Generator frequency

Bit	Status register value
8-0	Readback control register
9	Overflow
10	Total power ready
15	Total power overflow
others	unused (set to 0)

Table 6: Control and status register interface

Bit 3-0 of the control register selects which channel output is sent to the monitor DAC output. This is useful to observe e.g. a converted sinewave on an oscilloscope.

Bit 4 allows a sine generator to replace the input signal. The sinewave frequency is set using bits 15-6 of the control register, with a step of 0.5 MHz.

Bit 5 enables the 1PPS sync circuitry. When it is set, the internal 1PPS signal locks to the rising edge of the incoming 1PPS. Once the circuitry is synced (usually by letting the enable high for more than a second), this bit must be reset, or the internal 1PPS will follow any jitter in the input 1PPS, and the number of clock cycles between successive pulses is not guaranteed to be always 128.000.000. When the bit is cleared, the internal 1PPS is generated from the ADC clock.

Status bits 8-0 simply reflect the corresponding control bits. They can be used for simple write/read checks of the interface. Bits 15, 10 and 9 are set when the specified event occurs (overflow or Total Power End of Integration), and are latched until read. They are automatically cleared by the read operation.

Registers 16-31 refer to the output channels $n-16$. The write register sets the threshold value for the *magnitude* bit in the hardware units. The read register reads the total power value, as a 32 bit signed value. The total power meter integrates the signal between successive 1PPS pulses, and bit 10 of the status register is set when results are available. Bit 15 is also set if an overflow has occurred in one or more Total power meters.

Total power units are such that the square root of the value read corresponds to the RMS of the signal in hardware units divided by 1.024. As the threshold optimal value for 2 bit quantization is 0.9076 times its RMS value, the threshold value to be written in the threshold register is the square root of the total power measurement multiplied by (0.9076/1.024). The corresponding statistics, for a Gaussian noise, is given in tab. 3.

4.2 Control software

The control software has been developed using an object oriented structure. A general programming interface for FPGA-based hardware has been developed for the ALMA based boards, and has been adapted for the dBBC.

The basic objects in this package are:

- `Cpld2Interface` A generic interface that deals with the details of the communication with the hardware (in our case with the parallel interface of the host computer). The name derives from the programmable design that implements this interface on ALMA correlator boards.
- `HardwareBlock`, a generic piece of hardware that includes several programmable registers. A `HardwareBlock` is instantiated specifying the `Cpld2Interface` used for communication, and an address, specifying a board ID and a block index. In this way up to 4 different hardware blocks can be hosted in the same FPGA. Each specific block (e.g. digital BBC, filterbank, spectrometer) is subclassed form this class.
- `Poly16` subclassed from `HardwareBlock`, implements a 16 channel polyphase filterbank

The `Poly16` class implements the following methods:

- `int TPRead(unsigned long results`) Read all the total power counters, in the array `results`, that must have i at least 16 elements allocated.
- `int SetGain(int gain`) Set thresholds using values specified in the array
- `int SetGain()` Set quantization thresholds performing a total power measurement and choosing the right threshold values
- `int Monitor(int chan)` Select output DAC monitor channel
- `int WaitTP()` Wait for Total Power data to become available
- `int SyncPps()` Sync 1pps circuitry to the external 1PPS signal

This class has been used to create a small control program, that every second checks the total power data, print them and dynamically adjusts the thresholds. It is listed in the code below.

```
int main(int argc, char* argv[])
{
    Cpld2Interface intf;         // Define the card, open it, etc
    Poly16 poly(&intf, 0x1000);  // Connect the object to card 1, chip 0
    unsigned long tpData[16];    // Total power data
    poly.SyncPps();              // Sync the internal 1pps to the input
    poly.SetGain();              // Set thresholds according to signal
    int j;
    bool quit=false;
    while  (true) {
poly.SetGain();          // dynamically adjust thresholds
        poly.WaitTP();           // Wait for TP results (every 1pps)
        int flag = poly.TPRead(tpData); // flag != 0 means overflow
        cout << (flag=0) << ``:'';      // Print results
        for (j=0; j<8; ++j) cout << `` '' << tpData[j]/1e6;
        cout << endl << ``  '';
```

```
        for (j=8; j<16; ++j) cout << ‘‘ ’’ << tpData[j]/1e6;
        cout << endl;
    }
    return 0;
}
```

References

[1] G. Comoretto, G. Tuccari: *Reference design for the Digital BBC Architecture*, Arcetri internal report 2/2008

[2] G. Comoretto, A. Russo: *Software di comunicazione con il correlatore Altera* Arcetri Internal Report 2/2007

[3] S. Winograd: *On computing the Discrete Fourier Transform* Proc. Nat. Acad. Sci. USA Vol. 73, No. 4, pp. 1005-1006, April 1976

[4] A. Russo: *Spectroscopic Instrumentation for Radioastronomy* PhD thesis, March 2009

[5] T. Toivonen*Number Theoretic Transform-Based Block Motion Estimation*, Department of Electrical Engineering, University of Oulu, Finland, Diploma Thesis, 2002

Contents