WHITE PAPER

Intel® FPGA

Versatile Channelizer with DSP Builder for Intel® FPGAs

Intel FPGAs enable high-performance wideband digital polyphase filter banks for modern communication and radar applications.

Authors

Dan Pritsker
Engineering Manager
Intel® Corporation

Hong Shan Neoh
Design Engineer
Intel Corporation

Colman Cheung
Design Engineer
Intel Corporation

Amulya Vishwanath
Product Marketing Manager
Intel Corporation

Tom Hill
Product Line Manager
Intel Corporation

Table of Contents

Introduction

The use of digital polyphase filter banks in modern signal processing systems has increased dramatically over recent years. This increased use stems from a desire to deliver more capable, innovative and accessible systems while sharing a common physical infrastructure. This is especially true when strict regulations on Radio Frequency (RF) spectrum use result in spectrum congestion. Systems engineers are attempting to push beyond the boundaries of existing practical limits by leveraging hardware platforms that deliver greater processing bandwidth to compensate for different adverse effects that become evident at higher frequencies.

Today's state-of-the-art systems use extremely high RF that operate using millimeter length waves. This dramatic expansion enables many unoccupied (aka."white") bands that system engineers can leverage to expand the operational bandwidth of their systems without interference. The ability to expand the operational bandwidth provides an opportunity to expand usage and features for almost every RF application. This is especially true for communication systems that use wideband to provide higher data rates over the channel as defined by Shannon theorem.

Shannon formula: $C=BW\cdot\log_2(1+SNR)$

Shannon channel capacity theorem dictates that if the system uses higher bandwidth it is possible to operate in much lower signal-to-noise ratio (SNR) specifications that results in a lower transmit power density. For communications systems, this translates to lower probability of intercept or jamming while also allowing these systems to have better resilience to a multi-path effect by operating with a shorter pulse duration for transmitted bits. Modern radar systems also benefit from the ability to operate at wider bandwidths by increasing resolution and the ability to discriminate two adjacent range reflections. Synthetic aperture radar (SAR) requires extremely wide bandwidth of the waveform to acquire terrain images with resolution of inches.

FPGAs provide an ideal processing platform for the implementation of polyphase filter banks for wideband communications and radar systems. Currently the largest available FPGA device includes over 10,000 high-performance mathematical blocks, randomly accessible memory and programmable routing that enable the use of multiple processing pipes that execute in parallel while maintaining overall power consumption efficiency. FPGAs are widely used to implement Digital Filter-Bank applications.

This technical white paper dives deep into capabilities of polyphase fast Fourier transforms (FFTs) for Intel® FPGAs.

Theory of Operations

A channelizer or an analysis filter bank takes a signal with certain bandwidth and divides it into several narrow band signals. Each output band or channel is a baseband signal itself. A channelizer achieves its goal by modulating (frequency shifting) the band of interest to baseband, filter out the unwanted bands, then decimate it to a narrow band signal, as shown in left hand side of Figure 1. Alternatively, the band of interest can be band-passed first with a complex filter, modulated to the baseband, then decimated to narrow band signal, as shown in the right-hand side of Figure 1.

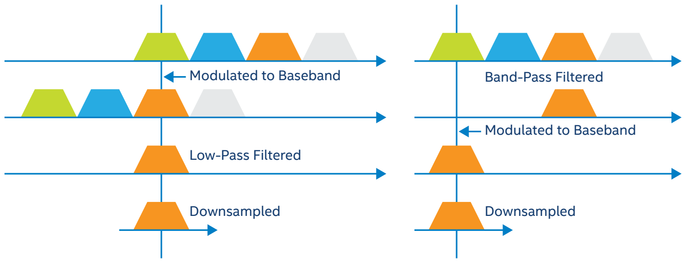

Figure 1. Channelizer Illustrations for Individual Channel

Polyphase Filter Bank

Instead of extracting each channel individually, all channels can be extracted very efficiently with polyphase filter bank (PFB) and FFT as shown in Figure 2.

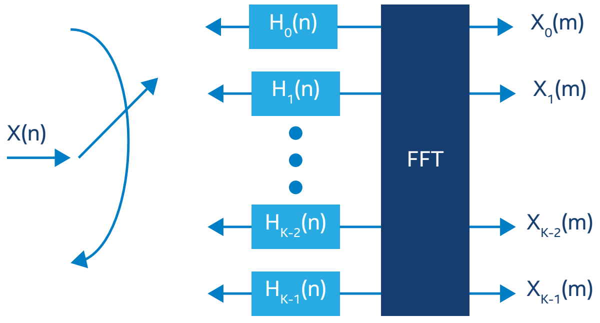

Figure 2. Block Diagram of PFB and FFT Channelizer

The details of the mathematical derivation of the FFT-based PFB channelizer can be referenced in [1], [2], and [3]. The intuition is offered below. To help explanation, let K be the total number of channels and k be the channel index range from 0 to K-1. Let p be the phase or branch index of the PFB, also ranging from 0 to K-1. Let T be the number of taps of coefficients per phase, therefore, the overall filter has K * T number of coefficients.

A polyphase filter (PF) is formed by breaking down a prototype low pass filter into K sub-filters by interleaving the coefficients of the prototype filters across K groups. For example, a prototype filter with taps [h0, h1, , , hT*(K-1)], can be arranged as K sub-filters as shown in Figure 3. These K sub filters still bear the characters of the original prototype filter, just K down-sampled. These sub filters are also 2π/K separated phase-wise.

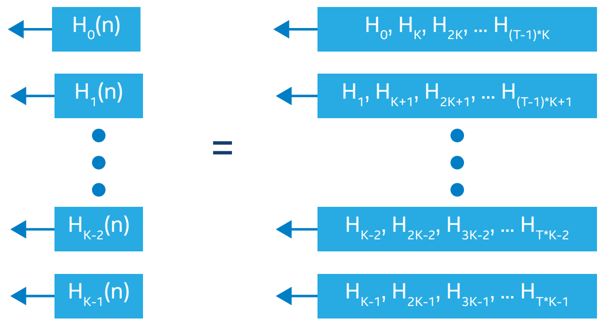
$H_0(n)$
$H_1(n)$
$H_{K-2}(n)$
$H_{K-1}(n)$
=
$H_0, H_K, H_{2K}, \dots H_{(T-1)*K}$
$H_1, H_{K+1}, H_{2K+1}, \dots H_{(T-1)*K+1}$
$H_{K-2}, H_{2K-2}, H_{3K-2}, \dots H_{T*K-2}$
$H_{K-1}, H_{2K-1}, H_{3K-1}, \dots H_{T*K-1}$

Figure 3. Formation of Polyphase Filter Coefficients

The selector on the left-hand side of Figure 2 also down sample the input by K:1 into each sub filter. At the output of the polyphase filter, there are K down sampled outputs with different phase information. Notice that the amount of information at the input and output of the PF has not changed, but transformed and bandwidth limited per prototype filter design.

If the outputs of the PF are just summed, only the channel at the baseband (0th channel, k=0) are preserved as shown in top-left of Figure 4. Information on all other channels are destructively canceled out. If each phase, p, of the PFB output are spun by W_K^{-p} before being summed, then the next (k=1) channel is extracted to the base band as shown in top-right of Figure 4. If the output of the PFB phases are spun by W_K^{-kp} before being summed, then the kth channel is extracted to the base band as shown in bottom figure of Figure 4.

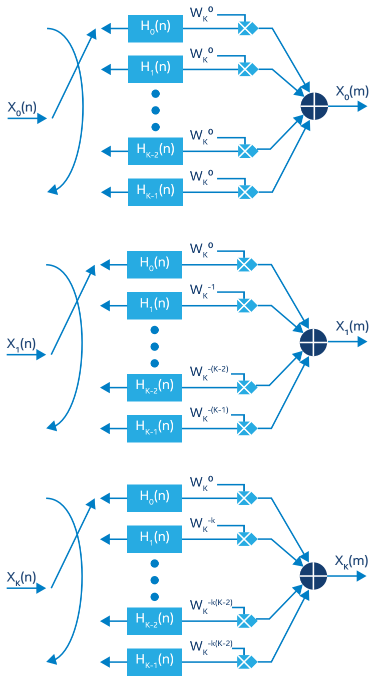

Figure 4. PFB Channelizer Mechanism for Channel 0, 1 and k

The outputs $X_0(m)$, $X_1(m)$, and $X_k(m)$ are the 0th, 1st, and kth output of a regular DFT or FFT. The FFT efficiently modulates all the PFB phase outputs into individual base bands signals. The amount of computation for modulation is reduced from K^2 to $K(\log_2 K)$. It should be cleared that the FFT operation here has nothing to do with time and frequency domain conversion, just a computationally efficient way to implement the frequency spinners W.

Oversampling Channelizer

There are variations of channelizers with different characteristics, the fore mentioned one (critically sampled channelizer) is the basic one where the bandwidth of each channel is exactly 1/K of the original signal spectrum. Naturally, the output baseband is also decimated by K. It is possible to design a channelizer so that the bandwidth of each channel, 1/M, is larger than 1/K of the original spectrum. In other words, the channel spectrum overlaps with its neighbors. This is a very useful class of channelizer and is called oversampling channelizer or overlapping channelizer, or Weighted Overlap and Add (WOLA) structure in [3].
Let M be the channelizer decimation ratio (between input sample rate and output sample rate of a channel). A critically sampled channelizer has M equals to K. An oversampling channelizer has M<K. Figure 5 shows the effects of oversampling channelizer by overlapping the input samples. The ratio K/M is commonly known as oversampling ratio – for every M inputs, there are K outputs (one for each channel).

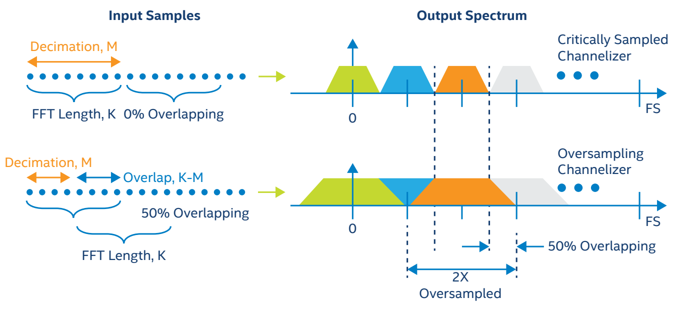

Figure 5. Input or Output Relationship of Oversampling Channelizer

To provide more output samples, inputs to the polyphase filter should be increased accordingly. This is done so with the decimation rate M less than K. Thus, there will be an K-M overlapping input samples to the PFB. Figure 6 shows the arrangement of data feeding into corresponding sub-filters for dot multiplications. Each column of orange boxes represents a group of polyphase filter input data that generate one column (or K) outputs. Notice that there are M data samples apart between two successive groups of input data, hence, decimation rate is M. For example, if M=K/2, the successive PFB inputs are only K/2 samples apart. The bottom half of the first data group overlaps with the top half of the second group. In such case, both PF and FFT should work twice as hard to produce 2X output samples.

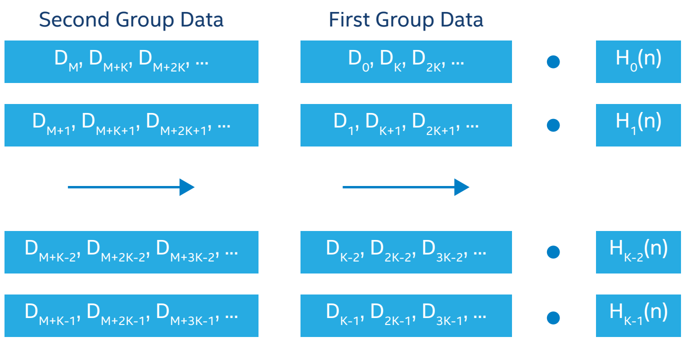
Second Group Data
First Group Data
$D_M, D_{M+K}, D_{M+2K}, \dots$
$D_0, D_K, D_{2K}, \dots$
$H_0(n)$
$D_{M+1}, D_{M+K+1}, D_{M+2K+1}, \dots$
$D_1, D_{K+1}, D_{2K+1}, \dots$
$H_1(n)$
$D_{M+K-2}, D_{M+2K-2}, D_{M+3K-2}, \dots$
$D_{K-2}, D_{2K-2}, D_{3K-2}, \dots$
$H_{K-2}(n)$
$D_{M+K-1}, D_{M+2K-1}, D_{M+3K-1}, \dots$
$D_{K-1}, D_{2K-1}, D_{3K-1}, \dots$
$H_{K-1}(n)$

Figure 6. Polyphase Filter Input Data Arrangement

There is one additional computation step when M is not equal to K. Referring to Figure 7 below summarizing the relationship between M and K, the modulation term W_K^{-kmM} becomes 1 when M=K, but evaluates to different values based on m and k. where m is the output index. This modulating term is applied after FFT output in PFB implementation.
The top portion of Figure 8 shows a complete oversampling channelizer structure.

One optimization is that modulation after FFT is the same as time delay (shifting input) before FFT. Doing so helps to reduce number of required multipliers, in the expense of buffering memory. Such buffering may become free if buffering is required anyway. The bottom portion of Figure 8 shows such a structure.

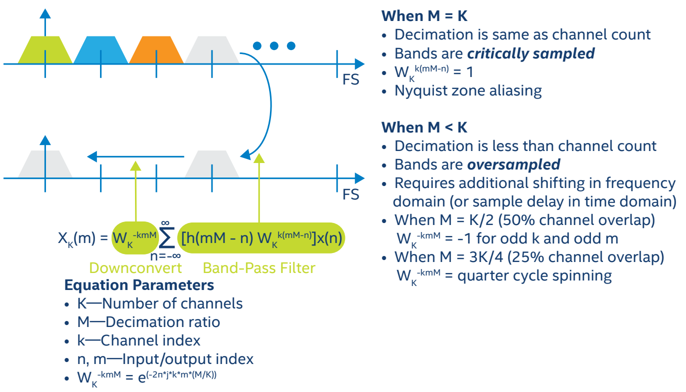
When M = K
• Decimation is same as channel count
• Bands are ***critically sampled***
• $W_K^{k(mM-n)} = 1$
• Nyquist zone aliasing
When M < K
• Decimation is less than channel count
• Bands are ***oversampled***
• Requires additional shifting in frequency domain (or sample delay in time domain)
• When M = K/2 (50% channel overlap) $W_K^{-kmM} = -1$ for odd k and odd m
• When M = 3K/4 (25% channel overlap) W_K^{-kmM} = quarter cycle spinning
FS
FS
$X_k(m) = W_K^{-kmM} \sum_{n=-\infty}^{\infty} [h(mM-n)\, W_K^{k(mM-n)}]x(n)$
Downconvert
Band-Pass Filter
Equation Parameters
• K—Number of channels
• M—Decimation ratio
• k—Channel index
• n, m—Input/output index
• $W_K^{-kmM} = e^{(-2\pi*j*k*m*(M/K))}$

Figure 7. Channelizer Characteristics per M and K Relationship

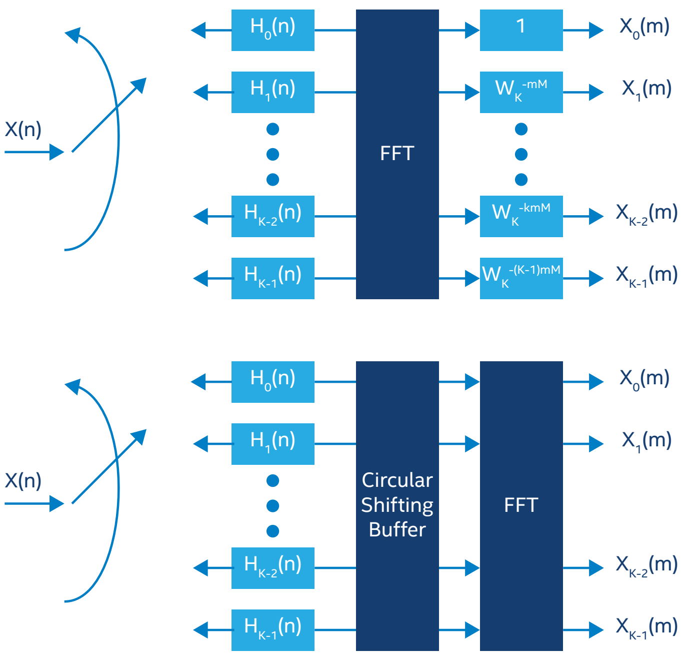

Figure 8. Oversampling Channelizer Structures

Filter Design

The prototype filter of a polyphase filter is basically a low pass decimation filter with bandwidth of 1/M. In oversampling channelizer, 1/M is larger than 1/K. A causal design may set the stop band to 1/M as in the top of Figure 9, so that the transition band is (1/M – F_{pass})/2. A more resource efficient design would double the transition bandwidth as shown in bottom of Figure 9, so that half of the transition band is within 1/M and the other half is outside. In other words, (F_{stop} + F_{pass})/2 equals to 1/M. The energy outside 1/M folds back to the transition band within 1/M, and the passband is not affected. It may nevertheless introduce slightly more noise from the larger transition band. Doubling the transition bandwidth roughly reduces the number of coefficients to half.

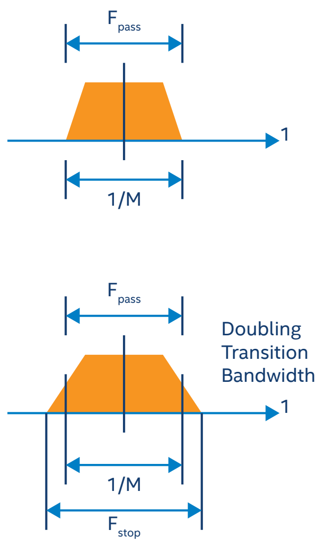

Figure 9. Low Pass Filter Design

Inverse Channelizer

Inverse channelizer or synthesis filter bank is just the reverse of the channelizer. Figure 10 shows the back-to-back channelizer and inverse channelizer. For x(n) at the output to be perfectly reconstructed from x(n) at the input, filter H(z) and G(z) has to be inverse of each other. or, $H(z)G(z) = cz^{-d} I_K$ where c and d are constants.

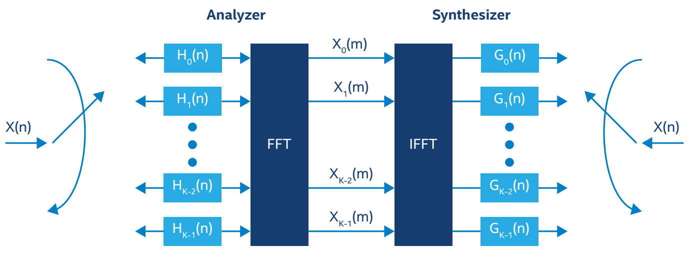

Figure 10. Back-to-Back Channelizer and Inverse Channelizer

Fixed-Point vs Floating-Point Numeric Precision

When implementing algorithms hardware developers will need to tradeoff between floating-point and fixed-point numeric precision. Algorithms are typically developed using double-precision floating-point numbers. This provides almost infinite dynamic range and infinitesimal resolution and allows the algorithm developer to focus on the mathematical part of the algorithm without being concerned about the effects of a reduced precision datatype on their algorithms. When implementing these algorithms in hardware, however, double-precision floating-point is expensive forcing the designer to evaluate reduced precision data type formats that typically include single precision floating-point or fixed-point. These data types can reduce the cost and power of the final system but add to the complexity of the design process by requiring analysis of the reduced numeric precision effects on the algorithm response.

A fixed-point number defines a dedicated number of bits for the integer and fractional parts of the number. No matter how large or small the number the same number of bits are used for each portion. For example, a 16-bit fixed-point number is typically defined using the nomenclature "unsigned (16,13)", which means this is an unsigned number that uses a total of 16-bits to represent the number where 13 of the bits are used for the fractional (right of the decimal point) and 3 bits are used for the integer portion of the number (left of the decimal point). If more numeric range is required than more bits can be allocated to the integer portion or if finer resolution is required more bits can be assigned to the fractional portion of the number.

In contrast, a floating-point number does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to indicate where within that number the decimal place sits (called the exponent). So, a floating-point number that took up 10 digits with 2 digits reserved for the exponent might represent a largest value of 9.9999999e+50 and a smallest value of 0.0000001e-49. An IEEE 754 double-precision floating-point number requires 64 bits and an IEEE 754 single-precision floating-point number requires 32 bits.

Converting an algorithm from floating to fixed point can be a complex and tedious process that requires an analysis of the effects of the reduced numeric precision on the algorithm performance but can yield hardware cost and power savings. Intel FPGAs support both floating and fixed-point numbers but historically FPGA have supported "soft" implementations of floating-point functionality that was implemented using fixed-point digital signal processing (DSP) blocks and other logic resources. Such implementation provided good numerical performance advantages but suffered from FPGA resource over-usage high-power consumption compared to fixed-point implementations. Intel Stratix 10 and Intel Arria® 10 FPGAs have solved this problem by incorporating IEEE 754 single-precision floating-point functionality into each DSP blocks. This makes the usage decision much easier as it does not imply any resources penalty with negligible power consumption difference versus fixed-point mode of DSP block.

Implementation

Polyphase Filter Implementation in DSP Builder for Intel FPGAs

The way K logical phases of a polyphase filter is arranged when designing with FPGA depends on the sample rate and the clock speed of the implementation. A comfortable clock rate for traditional FPGA signal processing is about 200 to 400 MHz. The phases can be time division multiplexed (TDM) in one datapath, in multiple parallel datapaths, or a combination of both. For example, if the input sample rate is 1024 Msps and there are 512 channels or phases, each channel only takes an output sample rate of 2 Msps. The most efficient implementation would be using four datapaths with each datapath carrying 128 channels. These variations of parallel paths and TDM paths can be conveniently implemented in the DSP Builder for Intel FPGA tool.

The DSP Builder for Intel FPGAs is a MathWorks* Simulink*-based tool with Intel FPGA specific components. The MathWorks Simulink tool natively handles vector signals and vector processing engines. Filter can be designed with primitive components or with built-in filter library model. For polyphase filter design, a custom library template was built using primitive components with configurable number of parallel paths, number of TDM channels and the prototype coefficients.

When the Simulink or DSP Builder for Intel FPGAs Advanced Blockset (DSPBA) runs, the polyphase filter is built at compile time. The number of parallel filter is built per the vector width. The corresponding coefficients (embedded with channel counts, taps-per-phase, and data type) are passed down to each parallel filter as shown in Figure 11, which also supports TDM channels and vector processing (for taps-per-phase).

The primitive filter uses streaming interface at the input and output. The streaming interface consists of 3 signals – valid (v), channel count (c), and data (d). The channel count indicates the TDM phase and selects the corresponding phase of 14 coefficients in parallel. The 14 coefficients are dot multiplied with a vector of 14 data and a scalar result is generated.

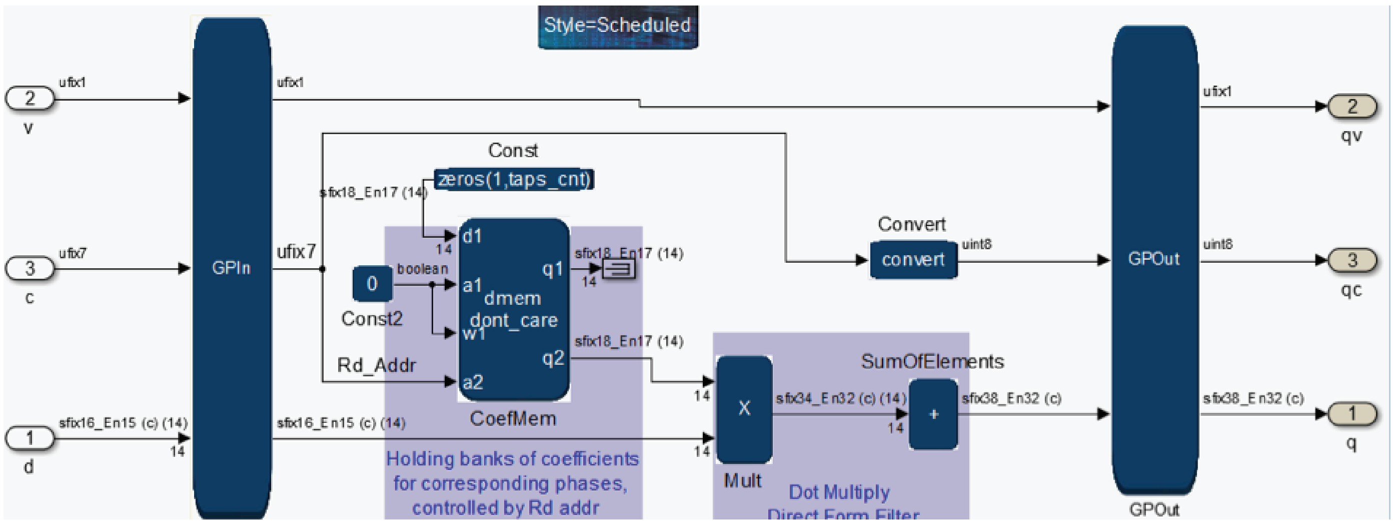

Figure 11. Polyphase Filter Implementation with DSPBA

Channelizer Forward FFT

In a standard super heterodyne structure, the input signal is mixed with a frequency provided by a local oscillator to translate to baseband. The complex multiplier in the super heterodyne structure is absorbed into an FFT in the polyphase channelizer architecture. This provides an efficient implementation compared to direct multiplication. The FFT is acting as spinners or phase rotators that efficiently convert each channel to baseband. Like the beamforming operation, it is constructively summing the selected aliased frequency components located in each path. Simultaneously, it is destructively canceling the remaining aliased spectral components. While the polyphase filter is performing the differential phase shifts across the same channel bandwidths, the FFT performs the phase alignment of the band center for each of the aliased spectral band. In the channelizer design, the FFT operation divides the input bandwidth into a number of evenly spaced frequency bands, commonly referred to as "bins", in order to allow the frequency content of an input signal to be analyzed. Instead of applying individual set of phase rotators to the filter stage outputs and summed to form each channel output, the discrete Fourier transform (DFT) operation can be used to simultaneously apply the phase shifters for all the channels required to extract from the aliased signal set. For computational efficiency, the FFT algorithm is used to implement the DFT. In terms of computation complexity and execution time, the polyphase FIR filter grows at the rate of O(N), while the FFT grows at the rate of O(N log N), where N is the number of channels. As the number of channel increases, the FFT dominates the overall execution. Hence, it is critical to have an efficient implementation of the FFT.

Direct implementation of the DFT requires on the order of N^2 operations to compute the channel values $y_0, ..., y_{N-1}$ for a single block of N data points. This is called the discrete Fourier transform. When $N = 2^i$ where i is a positive integer, and taking advantage of the intermediate values, the number of computations can be reduced to the order N log N operations. Hence, the name fast Fourier transform.

Polyphase FFT

For polyphase FFT with multiple parallel phases, the FFT decomposition breaks the FFT into smaller building blocks based on the number of phases, and FFT size. This is similar to the decomposition that is shown with the Cooley-Tukey algorithm as shown in the following equation.

$$X(k) = \sum_{n=0}^{N-1} x(n)W^{kn}$$

$$X(N_K k_1 + k_2) = \sum_{n_1=0}^{N_1-1} \left(W_{N_1}^{k_1 n_1} \sum_{n_2=0}^{N_2-1} x(N_1 n_2 + n_1)\, W_{N_2}^{k_2 n_2} \right) W_N^{k_2 n_2}$$

Where $N = N_1 + N_2$

The Radix-2^2 algorithm as referenced in [4] is a very efficient hardware architecture where pipelined feedforward structures uses separate hardware for each stage. This allows streaming data to be processed continuously in contrast to block-based FFT which processes the data in-place and requires separate buffers.

Radix-2^2 is based on Radix-2 and the flow graph of a Radix-2^2 DIF FFT can be obtained from the graph of a Radix-2 DIF one. The main difference is the twiddle factors. As proposed by Garrido, in the Radix-2^2 case, the twiddle factors have been changed into a trivial rotation in odd stages, and a non-trivial rotation in the successive stage as shown in Figure 12. This results in one half or less of the stages require complex multipliers for butterfly rotations, reducing the computational requirement.

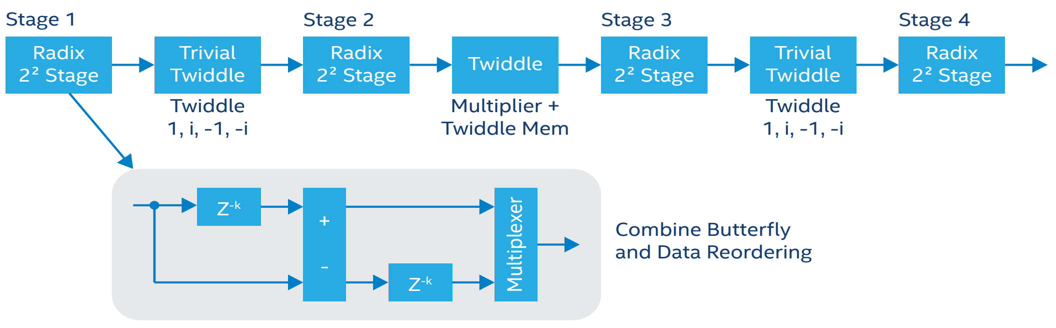

Figure 12. Radix-2^2 FFT Structure

The Radix-2^2 structure inherently supports parallelism. Figure 13. shows a 16-point 4-parallel radix- feedforward FFT architecture. The architecture is made up of Radix-2 butterflies (R2), non-trivial rotators, trivial rotators, which are diamond-shaped, and shuffling structures, which consist of buffers and multiplexers. The lengths of the buffers are indicated by a number.

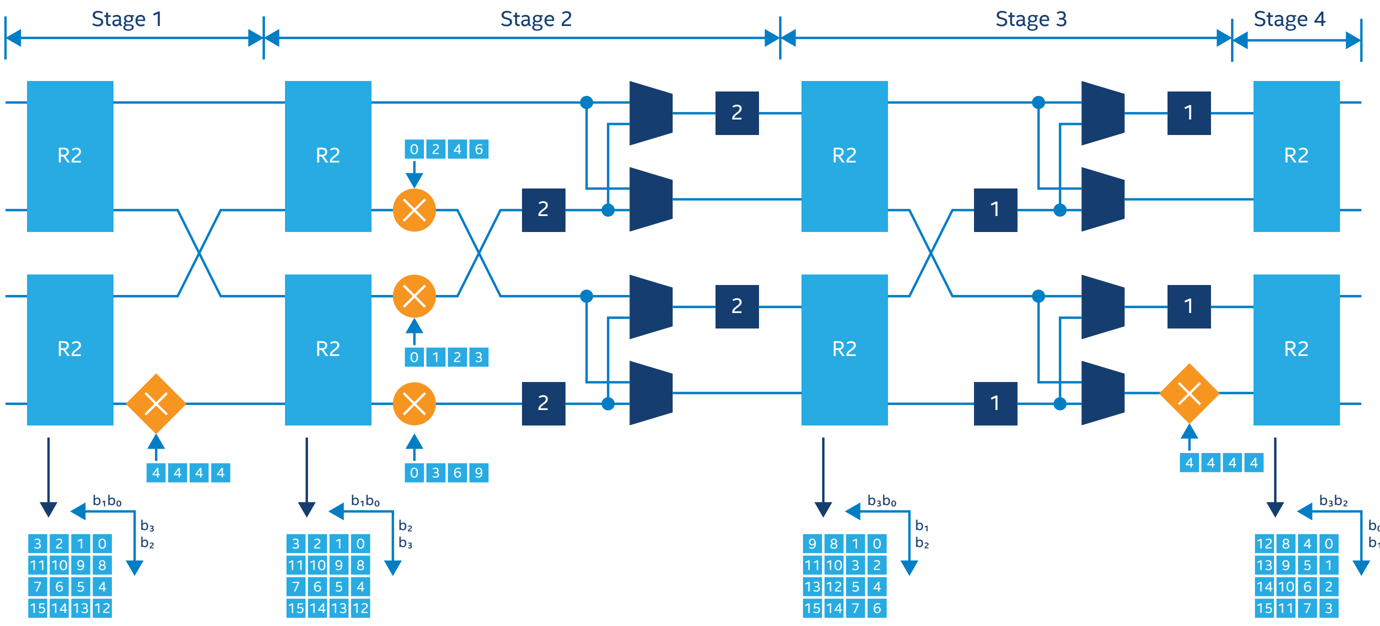

Figure 13. 4-Parallel Radix- Feedforward Architecture for the Computation of the 16-Point DIF FFT

However, it can be seen that as the parallelism of the parallel radix 2^2 topology continues to increase, the degree of coupling between the FFT paths will grow increasingly complex, and the connections difficult to route and close timing. This has resulted in hybrid type FFT architecture, leveraging the radix 2^2 concepts, but combining serial and parallel FFTs structures to create a scalable, high performance and resource efficient implementation.

The serial FFTs are more memory efficient. The single-path delay feedback architecture stores the butterfly output in the feedback shift registers where the feedback delays essentially stores half the elements at each stage. This makes the memory footprint very minimal. For fully parallel FFT using a split-radix architecture, it takes advantage of constant multipliers optimization and uses fewer overall multiplications.

Serial FFTs can be combined with parallel FFTs to produce hybrid FFTs. In the case of handling input with multiple phases, the serial FFT section consist of multiple single-wire serial FFTs in parallel. There is naturally a twiddle block between the two sections between serial and parallel. The parallel section is more multiplier efficient but less memory efficient than the serial section. To take advantage of both architectures, the serial FFT section represents the early stages in the FFT where the shuffling structures have longer feedback buffers, and the latter stages are implemented using a parallel FFT section. This provides a good trade-off between memory and multiplier resource usage. Figure. 14 shows a 16-path hybrid 512-K point FFT where the first six stages are implemented using a serial FFT section and the latter stages are implemented using a parallel FFT section with the twiddle stages in-between.

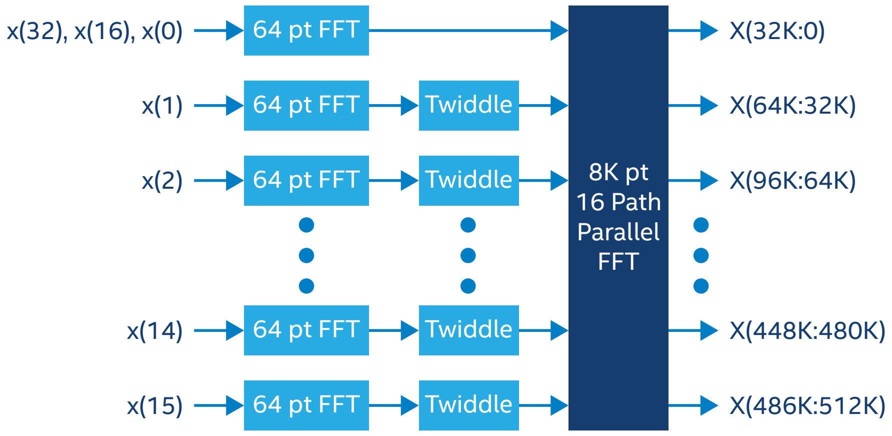

Figure 14. Architecture of an 512-K Point Hybrid FFT with Serial and Parallel FFT Sections

The FFT intellectual property (IP) block within the model-based DSP Builder for Intel FPGAs tool flow, allows full parameterization of the core from FFT size, number of parallel phases, number of serial/parallel stages, as well as twiddle factor implementation options to trade-off resource between logic and dedicated DSP blocks. The core also supports both fixed-point and single precision floating-point data type. For the fixed-point configuration, there are various pruning strategies to allow SNR performance and resource usage trade-off.

Real FFT

There are many cases where the inputs to the PFB is real instead of complex. For example, when the input follows directly from an analog-to-digital converter (ADC) with real only output. FFT is natively complex, a simple way is to set the imaginary part of the data to zeros and proceed with FFT. Half of the FFT outputs can be ignored as real input produces mirrored spectrum.

It is possible to take advantage of the unused complex input and perform the transform with a half-length FFT and additional processing. The steps are outlined below. Please see [2] for more details.

1. *g* is a real value sequence of 2N points and is mapped to an N point complex FFT, so that
   - $x(n) = x_1(n) + jx_2(n)$ where $x_1(n)=g(2n)$ and $x_2(n) = g(2n+1)$
2. Perform FFT Rearrange, with help of x*(n)
   - Transform $x(n)$ to $X(k)$,
   - with help of $x^*(n)$: $x_1(n) = ½ [x(n) + x^*(n)]$, $x_2(n) = ½ [x(n) - x^*(n)]$
3. After FFT, find X(k) and X(N-k)
   - $X_1(k) = ½ [X(k) + X^*(N-k)]$ $\quad X_2(k) = ½ [X(k) - X^*(N-k)]$
   - time reverse of $X(k)$ *is* $X(N-k-1)$, *not* $X(N-k)$ for $k=0,1,,,,N-1$
   - When $k=0$, $X(N-k) = X(N) = X(0)$. Thus $X(N-k)$ is shifted time reverse.
4. Forming G(k) as a function of $X_1(k)$ & $X_2(k)$
   - $G(k) = \sum_{k=0}^{N-1} g(2n)W_{2N}^{2nk} + \sum_{k=0}^{N-1} g(2n+1)W_{2N}^{(2nk+1)k}$
   - $= \sum_{k=0}^{N-1} x_1(n)W_N^{nk} + W_{2N}^{k} \sum_{k=0}^{N-1} x_2(n)W_N^{nk}$
   - $G(k) = X_1(k) + W_{2N}^{k} X_2(k)$, and $G(k+N) = X_1(k) - W_{2N}^{k} X_2(k)$ $\quad k=0,1,,,,N-1$
   - Only G(k) is needed, G(k+N) is the mirrored output at negative frequency

Figure 15 shows the computation chain implementing the real FFT. This process also works with parallel FFT, albeit is slightly more complicated.

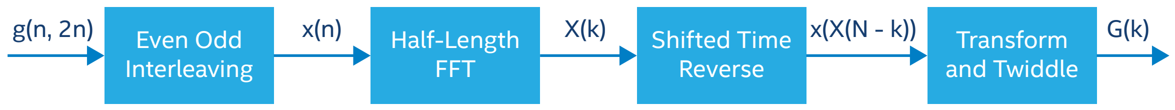

Figure 15. FFT-Optimized for Real Input

When FFT size is large, this approach may not be very efficient as the shifted time reverse operation needs additional double buffer memory. Latency is also increased by the FFT length. For large FFT or latency sensitive applications, it may be simpler by converting the real input to complex with the same signal bandwidth first, then proceed with a half-length FFT. This can be achieved efficiently by a quarter-band shift and a 2:1 decimation half band filter.

Oversampling Channelizer Structure

As mentioned in the theory of operations that oversampling channelizer is characterized by M, the decimation rate, being less than K, the number of channels. Depends on how much M is smaller than K, we may have two different architectures. First, if K is slightly more than M, say by 20%, then the polyphase filter and the FFT can just run 20% faster to provide the computation bandwidth. Second, when K is about twice that of M, the amount of computations doubles relative to the case M=K. Therefore, the processing pipe can be doubled while keeping the same operating clock.

It may not be trivial to design the data buffer to provide the data sequences as shown in Figure 6. Figures 16(a) and 16(b)show conceptual schematic to generate overlapping data for TDM equals to 1 and S. Control logics is not shown. T is the number of taps per phase. New data is represented by blue wires while old data is represented by brown wires. When TDM=1, all K channels are parallel paths. Input data at lower rate is buffered with a FIFO. The M new data, together with M*(T-1) + (K-M)*T old data provides the necessary data for filtering.

When there are multiple TDM slots, S, K/S phases or channels are handled every clock cycles. Some cycles may not use new data at all. Consider oversampling of K/M, for every M inputs, there should be K outputs, therefore, there are (K-M)/S cycles that only old data from memory is used.

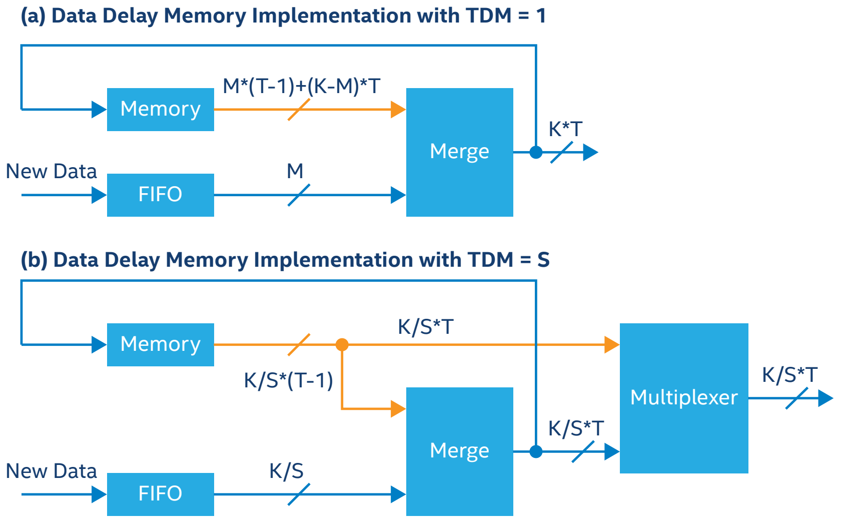

Figure 16. Data Delay Memory Implementation with TDM = S

It may also not be obvious that in the case of dual processing pipes (K is about 2*M), one data delay Memory may serve both polyphase filters as shown in Figure 17. Doing so saved quite a bit of memory.

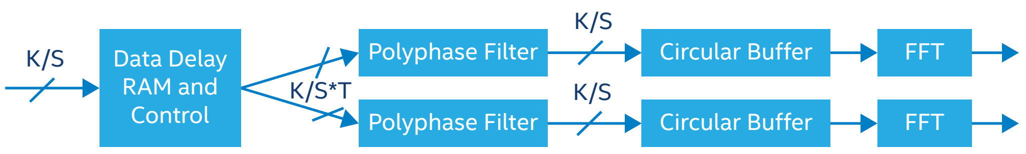

Figure 17. Oversampling Channelizer with Dual Processing Pipes

Inverse Channelizer – Inverse FFT

The inverse channelizer, also referred to as the synthesizer, reconstructs the frequency domain component from the channelizer back to the time domain sequence, to reconstruct a wideband signal. This is done using an inverse FFT. The inverse FFT structure is similar to the forward FFT block used in the channelizer. A simple approach is to swap the real and imaginary parts as shown in Figure 18.

$$F^{-1}(x) = S(F(S(x))) \text{ where } S(x) = im(x) + j.re(x)$$

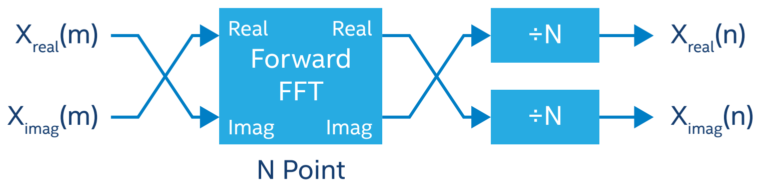

Figure 18. Compute Inverse FFT Using Forward FFT Architecture by Swapping Real and Imaginary Parts

Conclusion

Intel FPGAs provide a custom hardware processing platform that delivers the high performance and parallel processing paths necessary to implement state of the art wideband digital polyphase filter banks for modern communications and radar applications. The DSP Builder for Intel FPGAs provides a Simulink-based hardware design environment that includes a library of Intel FPGA highly optimized DSP building blocks including FFT that allows the creation of the custom Radix-2 parallel and serial FFTs required to achieve high performance with hardware efficiency. The DSP Builder for Intel FPGAs also enables rapid prototyping on supported hardware platforms.

References

[1] F. J. Harris, Multirate signal processing for communication systems, Upper Saddle River, NJ; Prentice Hall, 2004.

[2] John G. Proakis, Dimitris G. Manolakis, Digital signal processing: principles, algorithms, and application, Upper Saddle River, NJ; Prentice Hall, 1996.

[3] Ronald E. Crochiere, Lawrence R Rabiner, Multirate digital signal processing, Englewood Cliffs, NJ; Prentice Hall, 1983.

[4] Garrido, J., Grajal, J., Sanchez, M. A. and Gustafsson, O. , Pipelined Radix-2k Feedforward FFT Architectures, IEEE Trans. VLSI Syst., 15, 1, 2013.

[5] Parker, M., Finn, S., Neoh, H.S., Multi-GSPS FFT Using FPGAs, IEEE NAECON and Ohio Innovation Summit, 2016.

[†] Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.