Time Delay Digital Beamforming for Wideband Pulsed Radar Implementation

Colman Cheung, Ronak Shah, Michael Parker
Altera Corporation
San Jose, CA
ccheung@altera.com, rshah@altera.com, mparker@altera.com

***Abstract*—Radar system has been using a hybrid of analog and digital beamforming. Sub-arrays of phase shifters in the analog domain are digitized at the sub-array outputs for further digital beamforming processing. Such system suffers from limited bandwidth and is unable to create more than one simultaneous beam. Digital time delay beamforming brings many benefits such as wide bandwidth pulse waveform and simultaneous beams at different angles, wave forms and frequencies. Such capabilities may have been deemed too expensive in the past, but is now within reach due to the high density of multipliers in the FPGA and introduction of transceiver based ADC (Analog to Digital Converter)/ DAC (Digital to Analog Converter). This paper will present the design and implementation of a fractional delay (FD) filter used in time delay beamforming with Altera FPGAs and tools. Other beamforming issues like wide bandwidth, adaptive beamforming and high density interconnect architectures for phased arrays will also be investigated.**

Keywords—Time delay; Beamforming; Nyquist filter; FPGA; Phased array; Cholesky; QR Decomposition;

I. Introduction

Beamforming requires multiple antennas working together to create constructive and destructive interferences among antennas which produce a specific beam pattern. By delaying the signals feeding the array antennas (according to the distances from the phase center), the direction of the wave front can be adjusted accordingly. Weights on the antennas can also be set adaptively to minimize power from undesired directions.

For narrowband signals, shifting the phase of a signal has the same effect of delaying the signal. Phase shifters based on such principles provide phase delays. Phase delays for broadband applications result in dispersion of the signal – different frequency components of the signal experience different time delays. On the other hand, the true time delay chip can delay the signal by discrete amounts of time, but suffer from limited resolution. Both phase shifters and true time delay perform beamforming in the analog domain at the RF frequency. Additional concerns with analog beamforming include insertion loss from signal paths and only one beam can be formed at a time unless the antennas are partitioned.

The Fast Fourier Transform method has been very popular in digital beamforming due to its highly efficient implementation. It works very well for narrowband, but far from ideal for broadband signals.

Full Time delay digital beamforming eliminates drawbacks from analog beamforming and provides additional benefits like simultaneous beams at different angles, wave forms and frequencies. Arbitrary segmentation of array and arbitrary shape of antenna pattern can also be achieved. These properties are critical to (multiple) time sensitive missions. For broadband signals, time delays with digital signal processing can produce very precise results. Such capabilities may have been deemed too expensive in the past but is now within reach due to the high density of multipliers in the FPGA and the robust high speed serial interfaces of ADC (Analog to Digital Converter)/DAC (Digital to Analog Converter).

Section II describes the design and implementation of an FPGA efficient FD (Fractional Delay) filter. Section III describes other challenges such as the wide bandwidth implementation, design methodology and adaptive beamforming. Section IV looks into the interconnect architecture of high density phased arrays.

II. Fractional Delay Filter

A. Background

Reference [1] analyzed thoroughly the theory and solutions of designing fractional delay filters. It established the fact that an ideal fractional delay filter (delaying D samples) has impulse response as

$$h_{ideal}(n) = \frac{Sin[\pi(n-D)]}{\pi(n-D)} = Sinc(n-D) \qquad (1)$$

where D is the desired delay of the output sequences and D can be fractional. This implies that convolution with a phase shifted ideal filter would produce a delayed output accordingly. This property can also be generalized to any filter.

Radar beamforming also has a requirement that a new angle, thus a new delay and a new filter has to be generated in real time quickly with high precision. The two popular choices for the FD filters are Lagrange Interpolation and Farrow structure. Lagrange Interpolation is not appropriate in our application as the pass band is only flat near DC. Signals have to be up-sampled about 4 times before we have a wide and flat enough pass band. It would be more appropriate for application that follows the filter with an up-conversion.

Filter based Farrow structure using polynomial approximation is chosen for this implementation for its precise control of spectrum, SNR (Signal to Noise Ratio) and amount of delay. The Farrow structure is also simple to implement.

B. *Basic Filter*

Radar signal at base band has a certain bandwidth and is usually coherent and requires complex signal. The sample rate may be 25% to 50% higher than the bandwidth. Such requirements form our basic filter design specification. For example, signal bandwidth is 200 MHz, or +/-100 MHz about DC, minimum sample rate will be 200 MHz. At 50% oversampling, sample rate is 300 MHz. Obviously we need some sort of filter with pass band set at 100 MHz.

One may resort to a regular low pass filter with stop band set at 150 MHz or the Nyquist frequency, like in figure 1a. A low pass filter would indeed work, but is far from optimal, because it doesn't fully utilize the transition bandwidth. An optimal filter should have pass band of 100 MHz, stop band of 200MHz and sample rate of 300MHz, like in figure 1b. Since the transition bandwidth doubles that of the first filter, the second filter only takes about half of the taps as the first one.

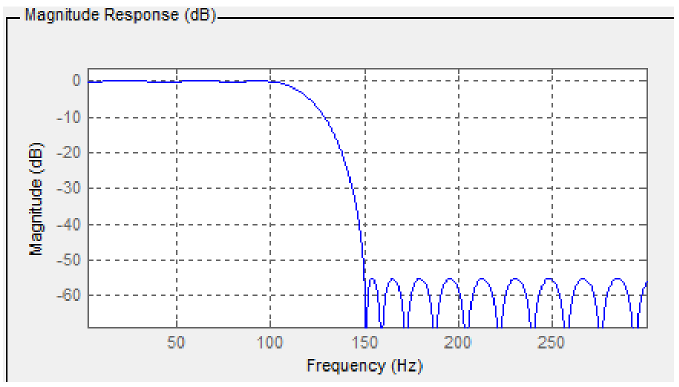


Fig. 1a. Low pass filter with Fpass/Fstop/Fs = 100/150/300Mhz

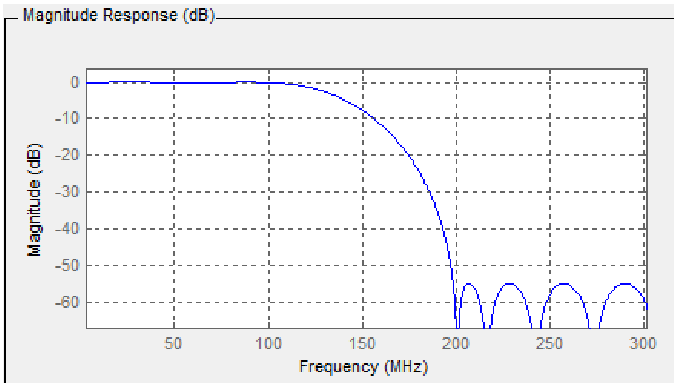


Fig. 1b. Nyquist filter with Fpass/Fstop/Fs = 100/200/300Mhz

Notice that the cutoff frequency of the second filter would then be at Nyquist frequency or 150 MHz. This is a Nyquist filter which is defined by

$$H(e^{j\omega}) = 1, \ |\omega| < (\pi / L)$$

$$H(e^{j\omega}) = 0, \ (\pi / L) < |\omega| < (\pi / L) \qquad (2)$$

When L=2, H is a half band filter. When L=1, H is a (full band) Nyquist filter perfect for resampling.

Please be aware that Matlab doesn't generate such filters as the stop band passed the Nyquist frequency. Figure 2a and 2b show such filter in blue. It is not much use by itself. The frequency response is 1 for all frequency. The impulse response has a one in the middle and all other coefficients are zeros. This is called all-pass FIR filter in [1] - obviously for its all-pass nature.

It is the interpolated or oversampled form that we are interested. Figure 2a and 2b show the interpolated version in red. The cutoff frequency of the interpolated version becomes a low pass filter and has cutoff frequencies at +/- 0.125 normalized. Recall that the impulse response of a rectangular filter of bandwidth B and time t is

$$h(t) = 2B * sinc(2Bt) \qquad (3)$$

Equation (2) shows that if the impulse response has maintained the same shape, a 4x interpolated or oversampled sinc function would have 1/4 normalized bandwidth. This explains why the frequency response of the 4x interpolated filter (red in figure 2a and 2b) shrinks to 25% of the non-interpolated one. The cut-off frequency is indeed at +/- 0.125 normalized frequencies. This implies that though we cannot design a Nyquist filter directly, we can design an oversampled version – which is exactly what we need.

The 4x oversampled filter coefficients can be seen as 4 phases of FD filter coefficients interleaved together as in figure 3. Any delay that falls in between can be interpolated from these pre-calculated phases. The more phases we have, the more accurate the represented filter would be.

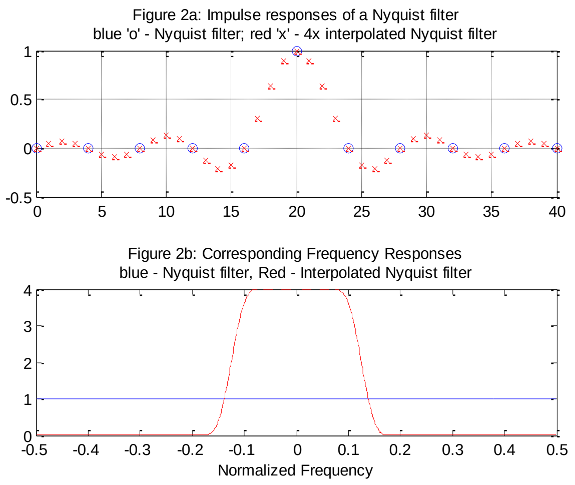


Fig. 2a, 2b Nyquist filter and its 4x interpolated version

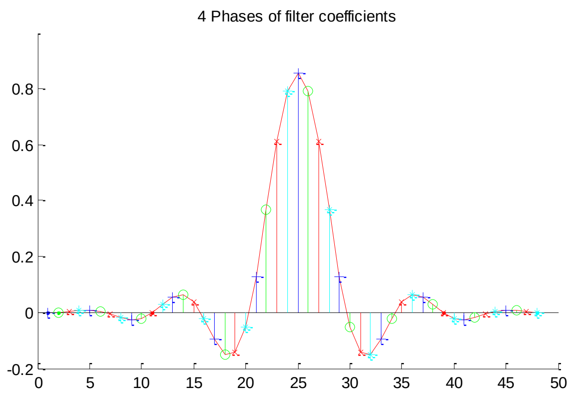


Fig. 3. 4 phases of filter coefficients

C. *Polynomial Approximation*

Equation (4) is provided by [2] to estimate the amount of coefficients oversampling, N, for a given data width of b bits, when linear interpolation between two neighbors are used.

$$N > 2^{(b-2)/2} \quad (4)$$

Derived from (4), table 1 shows the relationship of common signal width and the minimum number of oversampling in order to maintain the same fidelity. Therefore, a 64x oversampling would provide enough fidelity for a 14-bit signal, or about 84 dB SNR.

Table 1: Relationship between signal width and minimum filter coefficients oversampling

b (signal width)	N (oversampling ratio)
16	128
14	64
12	32

This is slightly different from the original concept of Farrow structure that one polynomial is used for each fractional delay filter tap. Here, many low order binomials are used for one filter tap. Depends on the delay, the corresponding binomial is selected.

D. *Implementation*

Although our implementation allows configurable number of antennas, bandwidth, sampling rate, number of filter taps and its oversampling ratio, the design requirements in table 2 are used for the purpose of illustrations.

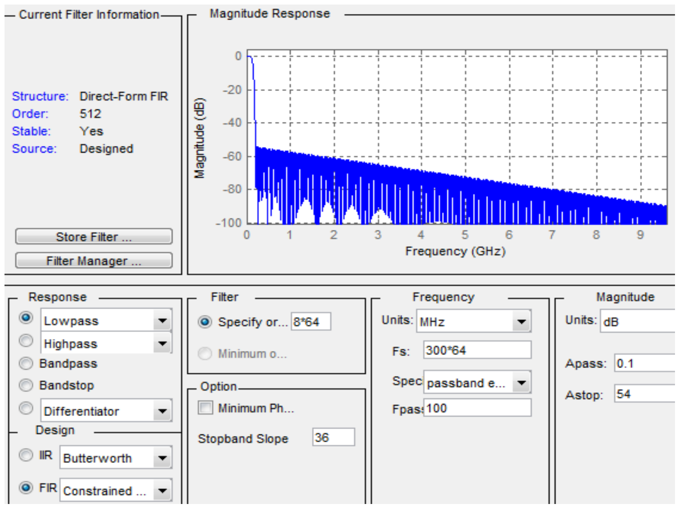
Table 2: Design specifications

Number of Antennas	32
Bandwidth	200MHz (double side band)
Sampling rate	300MHz
Number of filter taps	8
Number of binomials per tap	64

To provide 8x64 or 512 binomials, 513 oversampled coefficients or a filter order of 512 is needed. Each binomial is represented by two neighboring coefficients. The first binomial is represented by coefficients 1 and 2, and the second binomial is represented by coefficients 2 and 3, so forth and so on. The first filter tap is then covered by binomials 1 to 64 which in turn are covered by coefficients 1 to 65. The second tap is covered by binomials 65 to 128 which are in turn covered by coefficients 65 to 129.

The filter characteristics would look like figure 1b. Figure 4a shows the parameter settings of such filter design using Matlab filter design tool (FDATOOL). To create an oversampling filter, sampling frequency, Fs, and filter order are first multiplied with the oversampling factor - 64 in this case. Second, make sure that the stop band characteristic meets the requirement. If not, adjust the attenuation or the filter order accordingly. The stop band slope is specified so that rejected and overlapping spectral images will not build up.

Figure 5 shows the behavioral implementation of generating fractional delay coefficients in Simulink with Altera DSPBA (Digital Signal Processing Builder Advanced) design tool. The desired delay can be decomposed into an integer and fractional portions of sample period. The integer delay is achieved by delaying through a FIFO (First In First Out) and is not shown here. The FD becomes the PhaseAddr and FractionalPhaseAddr. For example, if FD is 0.2, or 12.8/64 (for 64 binomials per tap), PhaseAddr is then 12 and FractionalPhaseAddr is 0.8.



Structure: Direct-Form FIR
Order: 512
Stable: Yes
Source: Designed

Response: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design: IIR Butterworth [✓ FIR Constrained ...]

Filter: [✓ Specify or... 8*64] Minimum o...

Option: Minimum Ph... (unchecked)

Stopband Slope: 36

Frequency — Units: MHz; Fs: 300*64; Spec: passband e...; Fpass: 100

Magnitude — Units: dB; Apass: 0.1; Astop: 54

Figure 4a: Oversampled filter design

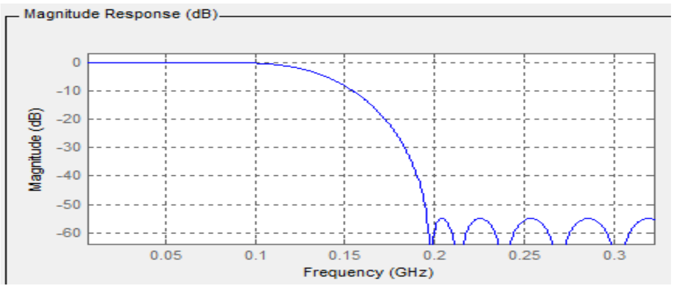


Fig. 4b. Oversampled Nyquist filter stop band characteristics

The oversampled coefficients are stored in memory or ROM (Read Only Memory) - SRC_CoefRom in the figure. Each memory has two read ports; one is addressed by PhaseAddr and the other one by PhaseAddr plus one. Therefore, both coefficients of the binomials are available at the same time. The final FD coefficient (of the corresponding tap) is interpolated between them. The "(32)" at the end of the data type means that the signal is a vector of 32 elements corresponding to 32 antennas. The vector size and data type are automatically propagated down the data path. Figure 6 (corresponding to SRC_scope2 of figure 5) shows that the 8 FD coefficients are calculated and loaded to the FIR in sequence, but with all 32-antennas calculated in parallel.

Figure 7 shows the convolution process through an array of filters. The "Array of FIRs" is a high level component parameterized to instantiate 32 independent FIR in parallel, each with a complex input port and a coefficient update port. The validity of the signal is indicated by the 'v' input and output ports. The scale block just scales the FIR output from 35-bit to 14-bit.

Although figure 5 and 7 shows the high level behavioral implementation, the design can be synthesized directly into hardware by the DSPBA tool. The right amount of pipelining registers will be inserted automatically at compile time depends on the operation clock speed, device type, vector or bus size, fan-outs and the circuitry.

This architecture is efficient as many phases of the FD filter can be pre-calculated. The real time FD coefficients only require simple interpolation. Implementation for such architecture is efficient for FPGA as FPGA block RAM (Random Access Memory) comes with size of 9Mbits or 20Mbits (depends on FPGA family), and can be configured in the form of 512x18-bits, or 1024x20bits. Though there are 513 symmetric coefficients, the synthesis tool is smart enough to recognize that coefficient 1 and 513 are the same (because the oversampled filter is symmetric) and automatically trim them down to modulo 512. Each antenna then mainly requires one block RAM and one multiplier for FD coefficients generation.

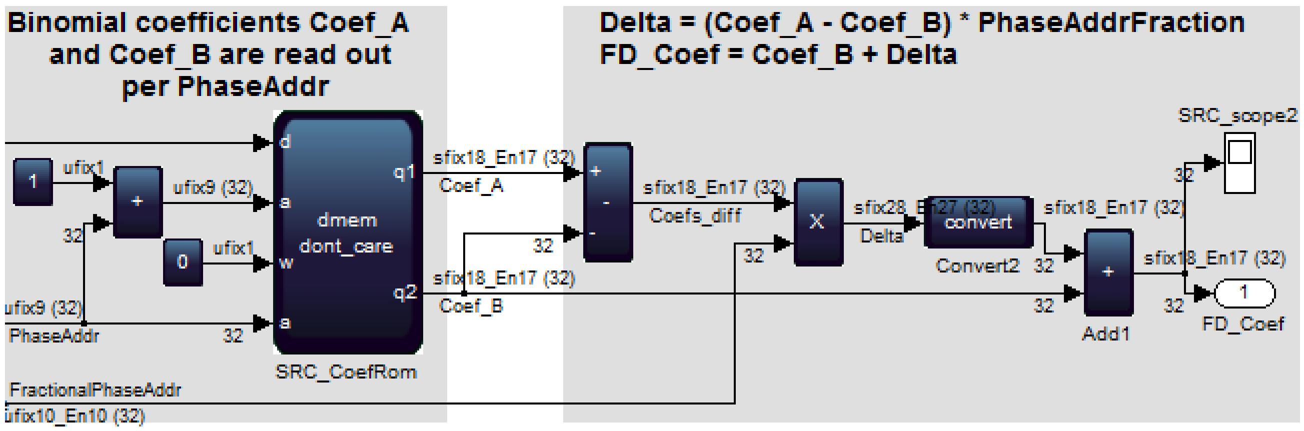


Fig. 5. Schematics showing Fractional Delay coefficients generation

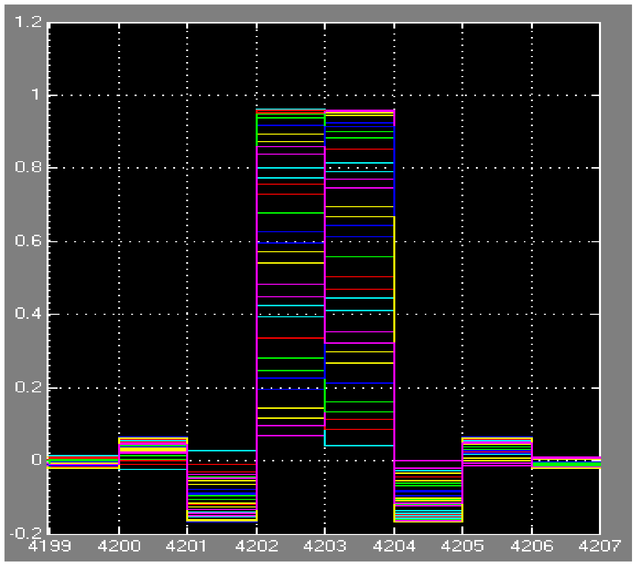


Fig. 6. FD coefficients update

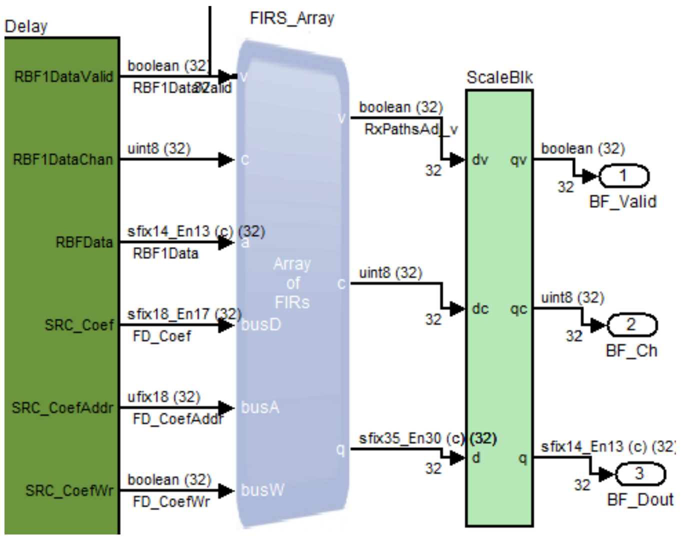


Fig. 7. FD Filtering and Scaling

The actual convolution with a complex signal takes 2 multipliers per filter tap. For an 8-tap filter and 32 antennas, it takes 512 multipliers per beam. This is why it is so important to use an optimal or near optimal filter. Table 3 shows the resource usages for 32 antennas per beam for different specifications. The usages also include other logics like converting beam angle to individual path delays. Large high-end FGPA in 28nm has about 4000 multipliers. Depends on the specifications, about 4 to 8 simultaneous beams can be implemented in an FPGA.

Table 3. Resources and Performances of different settings

	Spec. #1	Spec. #2	Spec. #3	Spec. #4
Sample Rate / clock (MHz)	300	300	267	250
Signal bandwidth (MHz)	200	200	200	200
Number of taps in FD filter	6	8	12	16
Expected SNR (dB)	40	52	60	65
Multipliers (18x18, max. 3926)	416	544	800	1056
Logics (max 260k)	8140	8800	9300	10k
Block RAM	64	64	64	64
Compiled Max. Frequency (MHz)	310	304	284	284

The precision of the beam depends on the delay precision of individual antenna which in turn depends on the phase calculation. The phase is calculated with a 16-bit address and sample period of 3.3ns. This gives us a resolution of higher than 0.1ps or signal travel distance of 0.03mm. Assuming antenna spacing is 0.5 lambda and lambda is 1m, angular resolution is then better than 0.005 degree. Mostly likely the bottleneck for the angular precision would be limited by other factors not analyzed here.

III. Other Implementation Challenges

A. System Level Design

Building DSP designs in HDL or FPGA usually involves two groups of engineers – system and hardware engineers. The system engineers implement the algorithms in Matlab or C and the hardware engineers take the specifications and build hardware to match the Matlab or C model. Simulink tries to bridge the gap with model based design and the FPGA vendors provide synthesizable block sets so that system engineers can turn designs into hardware. One big obstacle of this methodology is that the engineers have to pipelining the designs manually. FPGA doesn't run as fast as ASIC and register pipelining is the technique to clock the design more often so that it can achieve higher operating speed. It can be a tedious process.

Altera DSPBA tool solved the problem by automatically inserting pipelining registers based on the device type, fan-outs, operating speed, bus size and the design. The designer can focus more on the algorithm development and implementation. The automatic propagation of vector size and data types implies that DSPBA based design is natively configurable at compile time – just need to set the vector size and data type at the input. The tool also supports fixed point and floating point.

Figure 8 shows the major components of this beamforming design for pulsed radar in DSPBA. It starts with waveform (chirp) generation, transmit beamforming, emulations of free space loss, target echo and receiver noise, receive beamforming, aperture tapering, and finally pulse compression. The whole chain was built so that it can be tested and verified in an FPGA development board. Please notice that transmit and receive shares the same time delay beamforming engine as pulsed radar does not transmit and receive at the same time.

The hardware is controlled by a Matlab script via an API (Application Programming Interface) and USB (Universal Serial Bus). The USB is the conduit of a system bus (called system console) that is an integral part of Altera tools for debugging and testing.

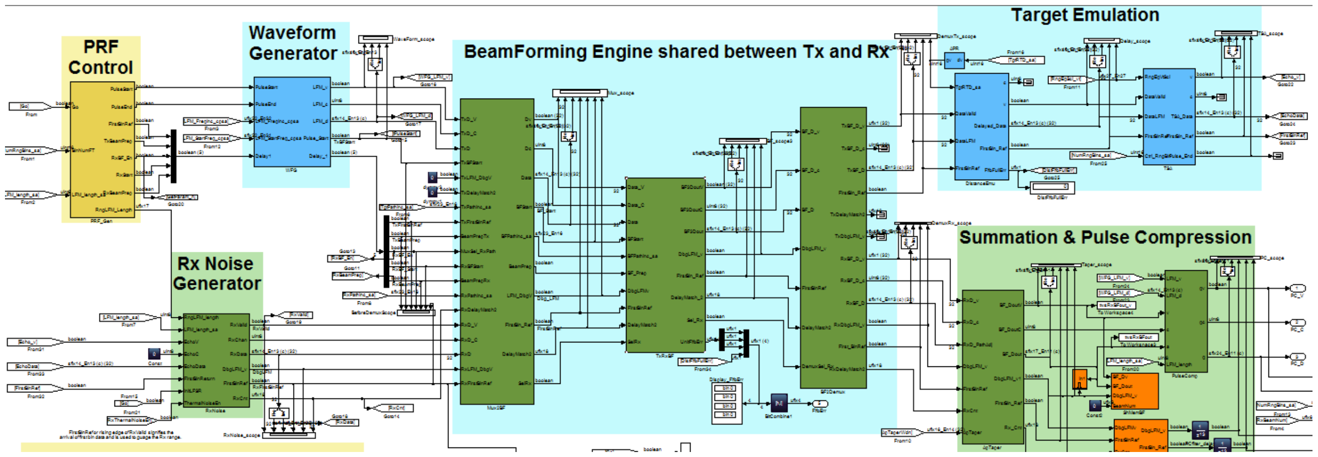

PRF Control
Waveform Generator
BeamForming Engine shared between Tx and Rx
Target Emulation
Rx Noise Generator
Summation & Pulse Compression

Fig 8. Top level design of the pulsed Radar beamforming

The script configures the hardware in real time with pulse waveform bandwidth and length, transmit beamforming angle, target location, and receive beamforming angle, etc. The Matlab script can also read the beamforming result back for analysis. Figure 9a and 9b shows the results captured in hardware and analyzed in Matlab. The receive beam is configured to scan between -60 and 60 degrees at 2 degrees resolution and find the target at -20 degrees and bin 100. The whole design and hardware verification was completed in 4 months with DSPBA. The development time would be at least double with separate system and HDL designs approach.

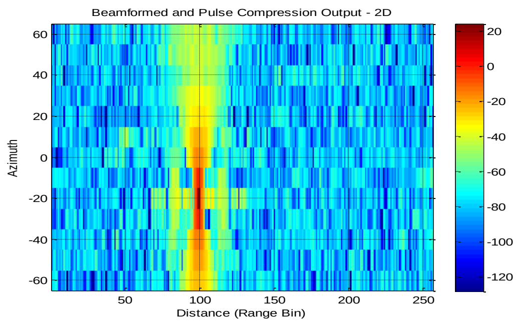


Fig. 9a. Time delay beamforming Output (Top view)

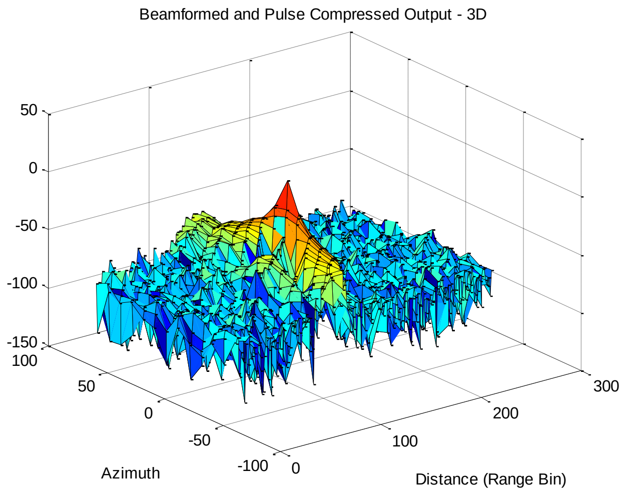


Fig.9b. Time delay beamforming output (3D)

B. *Wideband Consideration*

As mentioned, the beamforming example was built with 200 MHz signal bandwidth and FPGA clock speed of 300 MHz. What if the signal bandwidth requirement becomes 400 or 600 MHz? The same FPGA cannot run at 600 or 900 MHz. The DSPBA tool has prepared for such circumstances that sample rate is higher than clock rate and classify it as Super-Sample Rate (SSR). Major DSPBA high level components like NCO (Numerical Controlled Oscillator), filters, and FFT (Fast Fourier Transform) support SSR. For filters, the tool checks the sample rate and clock rate, if sample rate is an integer multiple of the clock rate, multiple parallel phases are generated. On the other hand, if clock rate is an integer multiple of the sample rate, the design is folded and multipliers are reused.

C. *Adaptive Beamforming*

Time delay beamforming alone is not sufficient for today's radar requirement. Modern radar must be able to handle jamming. Adaptive beamforming takes into account the effect of the jamming signal and tries to notch out the interference. The most popular method is SMI (Sample Matrix Inversion) and is based on the principle of match filtering. To do so, the interference (alone) is sampled or retrieved from the radar data cube, and the adaptive coefficient, h, can be found by

$$h = (k)\,(S^{-1})\,(t^*) \qquad (4)$$

where S is the interference matrix, t is the steering vector or the look direction, and k is normalization constant.

To find h, S must be inverted and is usually accomplished by Cholesky or QR decomposition. Both require floating point for stability (due to its capability to handle small numbers). Cholesky decomposition that works on covariance matrix also needs a wide dynamic range that floating point offers. Single Precision floating point offers 24-bit mantissa (including the implicit '1') and 8-bit exponent and has well enough precision and dynamic range to keep accumulated error low. Both Cholesky and QR decompositions have been built with the Altera DSPBA floating point, and may be integrated with the beamforming design in the future. Cholesky decomposition is simpler and achieves higher decomposition throughput. Table 4 shows the performance of Cholesky decomposition with different matrix size and vector size. Vector size refers to the number of processing elements in parallel. Number of channels refers to the number of matrices decomposed in time interleaved fashion.

Table 4. DSPBA Cholesky Decomposition Performance

Matrix Size	20x20	50x50	100x100
Number of channels	50	32	16
Vector size	20	50	50
Throughput (Matrices per second)	1M @ 255MHz	162K @ 225MHz	33.8K @ 225Mhz
Latency (us)	91.3	380	758

GPU (Graphic Processing Unit) and GP-GPU (General Purpose - Graphic Processing Unit) have been used in radar back end processing like adaptive beamforming and STAP (Space Time Adaptive Processing) partly because of its floating point capability. However, GPUs consumes much more power than FPGA. The physical multipliers in FPGA today is still fixed point, but used to implement floating point. This would be very inefficient if every floating point processing is first converted to fixed point (de-normalization), processed and then converted back to floating point (normalization), like in GPU operations. Due to the availability of 27x27 and 36x36 multipliers and the luxury of analyzing the whole processing chain at compile time, the DSPBA tool looks for opportunities where using larger mantissa widths can reduce the need of normalization and de-normalization. When needed, the normalization and de-normalization can also be implemented efficiently by multipliers instead of wide barrel shifter built from primitive logics.

IV. INTERCONNECTION

One problem that sub-array addressed is to reduce interconnection complexity between antennas and the signal processing units by grouping multiple antennas in one sub-array. However, to maximize the benefits of digital beamforming, each antenna should have its own ADC/DAC. ADC/DAC has had parallel interface and therefore limits the number of ADCs/DACs connecting to an FPGA. JESD204B, a serial interface standard for data converters established in July, 2011 allows raw bandwidth of up to 12.5 Gbps or payload up to 10Gbps using a single pair of differential signal. 16-bit ADC at 250Msps with 6.25Gbps interface is now available. A JESD204B link allows multiple devices and multiple lanes. The standard also provides means of synchronization across the link. Although replacing a phase shifter with a pair of ADC and DAC may be more expensive now, the price gap should be reduced in the future. The proliferation of wireless market also drives up the performance and lowers the cost of such devices. Table 5 shows the maximum number of transceivers and multipliers in the 28nm and 20nm Altera FPGA. All transceivers in these FPGAs can handle much more than the 12.5Gbps rate.

Table 5. 28nm and 20nm FPGA transceivers and multipliers count

Device	Number of transceivers	Number of multipliers
Stratix5 (28nm, high end)	36	3926
Arria10 (20nm, mid-range)	96	3356

With the serial interface each ADC or DAC requires only 2 pins and routing traffic management is then much simpler. Printed circuit board technology can control impedance down to +/- 5% to reduce insertion and transmission losses. Trace length can easily go over 50cm. Beamforming applications require trace delays from different ADCs/DACs to FPGA to be matched closely. Instead, they can be compensated in the time delay calculations by adding another (trace compensation) term for each antenna. Doing so also allows real time calibration of the entire antenna system.

To handle hundreds or thousands of antennas, a hierarchical approach of two or more layers of FPGA can be employed to distribute the transmit signals, and to aggregate the receive signals. Figure 10 shows an example of 1024 antennas through 2 layers of FPGA depicting the receive direction. Transmit path is similar, but having the direction reversed. The first (top) layer acts as a layer of (digital) sub-arrays and each sub-array has 32 antenna inputs and N outputs. Each output is a partial sum of certain beamforming angle. The second layer combines the partial sums and formed the final beam outputs. Therefore, the FD filter logics are only needed in the first layer, and the number of simultaneous beams, K, would be limited by the number of multipliers in the FPGA. It is obvious that N should be less than or equal to K, A beam may just need a subset of antennas and if the 1024 antennas are separated into M partitions. The maximum number of simultaneous beams would then be M*N.

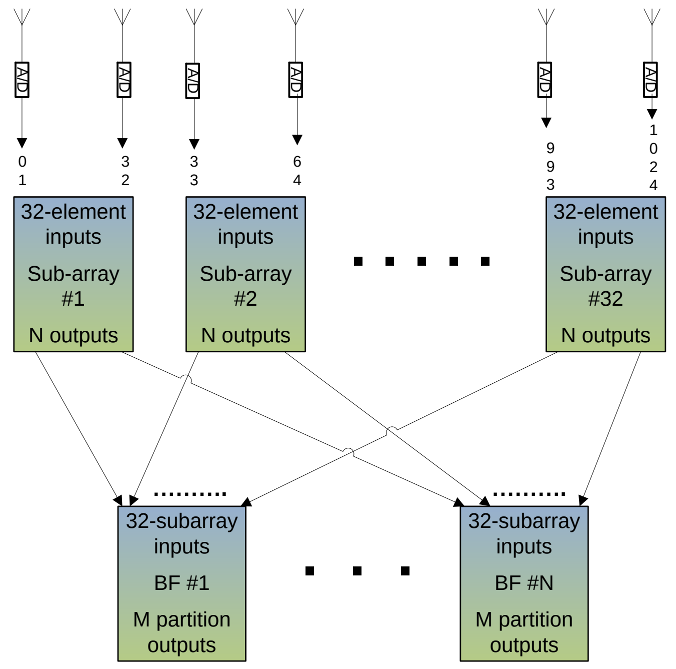


Fig. 10. Interconnection architecture for dense phased array

V. CONCLUSION

A practical design and implementation of an optimal FD filter in the context of time delay beamforming has been described. Related issues like design methodology, bandwidth flexibility, floating point requirement for radar backend processing are also investigated. Having the right design tool is paramount to complex signal processing designs and short design cycle time. Altera DSPBA was designed to deal with these challenges and will continue to evolve in this direction. Finally, the interconnection architecture for high density phased array was explored. All necessary technologies and components for time delay beamforming are available today and they may steer the evolution of radar designs.

REFERENCES

[1] T.I. Laakso, V. Valimaki, M. Karjalainen, U.K. Laine, “Splitting the unit delay: Tools for fractional delay filter design”, in IEEE Signal Processing Magazine (1996) 30-60

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

[3] M. A. richards, J.A. scheer, and W. A. Holm, Principles of Modern Radar: Basic Principles, Raleigh, NC: SciTech Publishing, May, 2010

[4] C.W. Farrow, “A continuously variable digital delay element”, in Proceedings of International Symposium on Circuits amd Systems, vol. 3, 1988, pp. 2641-2645,.

[5] M. Parker, “Radar Processing: FPGAs or GPUs?”, San Jose, CA: Altera Corporation, May, 2013

[6] S. Demirsoy, and M. Langhammer, “Fused datapath floating point implementation of Cholesky decomposition”, High Wycombe, UK: Altera Corporation, 2009

[7] “DSP Builder Handbook, Vol3: DSP Builder Advanced Blockset”, San Jose, CA: Altera, May 2013

[8] “An Independent Analysis of Altera’s FPGA Floating-point DSP Design Flow”, Berkeley, CA: Berkeley Design Technology, Inc., 2011

[9] "An Independent Analysis of Floating-point DSP Design Flow and Performance on Altera 28-nm FPGAs", Berkeley, CA: Berkeley Design Technology, Inc., 2012