FPGAs Make a Radar Signal Processor on a Chip a Reality

Ray Andraka, P.E., Andraka Consulting Group, Inc
Andrew Berkun, Jet Propulsion Laboratory

Abstract

Radar signal processors heavily tax the capabilities of conventional microprocessor based signal processing systems. Higher performance systems using custom silicon cost too much for the typically small production volumes, and are not flexible enough for research applications. Field programmable gate arrays offer the performance of custom silicon while maintaining the economies and flexibility of the microprocessor based solutions. Recent devices possess the density and performance to realize a complete radar signal processor in a single FPGA, including complex down-conversion of the IF to base-band. Performing the down-conversion and matched filtering in the FPGA eliminates the specialty chips previously needed for these functions, making it possible to use commercially available FPGA boards. Our system packs 4 channels into a pair of FPGAs. Each channel (a demodulator, a matched filter and a Doppler pulse pair processor) executes over 5 billion multiplies per second.

1. Introduction

Radar works by bouncing electromagnetic energy off a target, recording the echo and making some useful observation from the data. A fundamental problem in radar is that the vast majority of the reflected energy does not make it back to the receiver. Much of the processing in a radar system is to improve the signal to noise ratio of the received signal. Our system is intended to measure the amount and velocity of rain in and below a cloud from an aircraft above the cloud layer.

Our radar uses pulse compression, that is to say we spread the transmitted pulse out in time and then process the received echo with a matched filter to de-spread it. We also band-limit the return and digitally demodulate the signal to a complex baseband. Finally, we measure and average the echo power and the doppler shift between successive echoes to obtain the desired measurements. Our system uses four channels with different frequencies and field polarizations to obtain a more accurate picture of the weather.

The net processing for each channel requires over 5 billion multiplies per second - well beyond the capabilities of traditional microprocessor solutions. The low production volume and the desire to be able to adjust the processing in the future has driven us to an FPGA based solution. Furthermore, by doing the demodulation and matched filtering in FPGAs rather than with specialty devices, we can use commercially available boards, thereby avoiding the cost, risk, and schedule of designing our own. Application of a few algorithmic tricks, careful FPGA floorplanning and the remarkable density and performance of the current FPGAs allows us to put all the processing for two complete channels into a single FPGA.

2. Doppler pulse pair radar basics

2.1. Pulse pair processing

The echo for each radar pulse is demodulated and digitized into a large number of complex samples (range gates in radar terminology). Each sample index corresponds to a specific time offset from the start of the radar pulse, so each sample represents the reflected energy at a specific range. The complex time series of samples at a given range gate can be processed with a Fourier transform to obtain the doppler spectrum of the echo at that range, from which the mean velocity and variance can be obtained. For a pulsed radar, it can be shown that the pulse pair algorithm provides a reliable estimate of the mean doppler frequency[1] at each range gate:

$$R = \frac{1}{N}\sum_{k=0}^{N-1} s_{k+1}s_k^*$$

where s_k are successive samples at a range gate and * indicates the complex conjugate.

Since this algorithm is simpler to compute than an FFT and is not biased by receiver noise, it is the algorithm of choice for most meteorological applications. We need to compute the average power along with the mean doppler shift to provide a reference and for measuring the total area of the raindrops at a given altitude. The average power at each range is:

$$R = \frac{1}{N} \sum_{k=0}^{N-1} \mathrm{Re} \mid s_k s_k^* \mid$$

The mean doppler frequency output for each range gate is a complex value. Its phase indicates how much the raindrops have moved from one echo to the next. The magnitude gives a weight to the phase value (stronger echoes count more than weaker ones). The difference between the average doppler magnitude and the average power measurement provides a measure of the 'spread' of the velocities in a set of echoes.

2.2. Pulse compression

Transmitted pulse power plays a large part in the signal to noise ratio of the received echoes. Higher power means better detection. The range resolution of radar is inversely proportional to the width of the transmitted pulse, so we desire to make the pulse as narrow as practical to obtain the best range resolution we can. Unfortunately, narrowing the pulse means that the peak pulse power has to be increased to keep the total pulse power constant. High peak powers present many practical problems in the transmitter design [2].

Instead, we can transmit a long coded pulse, and then compress the echo into an impulse in the receiver using a matched filter. The idea is to spread the transmit energy out over time to limit the peak transmit power, then use correlation at the receiver to recover the range resolution. In our case, the coded transmit pulse is a linearly swept frequency 'chirp'. The transmit pulse can be viewed as a convolution of an impulse and a filter with an impulse response equal to the desired transmit waveform. Ideally, the receive filter is the inverse filter so that filtering the transmit waveform restores the original impulse. The impulse response of this receive filter is the complex conjugate of the time-reversed chirp. In practice we need to window the filter to suppress range sidelobes [3] and we will apply corrections for non-linearity and noise in the transmitter and receiver. We desire to do the matched filtering in the digital part of the processing to give us the flexibility of changing the transmit waveform and frequency, optimizing the filter for a particular doppler range, and applying corrections for the analog front end. The matched filter's reference waveform is as long as the transmit waveform, which in this case is 40 μsec. In order to keep the number of taps in the filter to a minimum, we operate the filter at the lowest sample rate possible. At 5 MHz, this translates to a minimum filter length of 200 complex taps. We've extended the filter to 256 taps to accommodate longer chirps in the future.

2.3. Demodulation and video filtering

Quadrature phase detection is used to obtain the real and imaginary parts of the complex echo envelope necessary for doppler pulse pair processing. By doing the demodulation from IF to complex baseband in the digital domain, we avoid the channel mismatch and drift problems associated with an analog quadrature phase detector. Additionally, using a digital video filter gives us a much better and more flexible filter than we can afford to build in the analog domain. The final analog IF is a 4MHz wide signal on a 5 MHz carrier. By oversampling at 20 MHz we can use an anti-alias filter with a fairly gentle roll-off, and as we will see shortly, we gain a significant processing advantage for the digital demodulation.

The digital video filter attenuates the noise outside the 3 to 7 MHz passband. That noise is mostly receiver thermal noise, so it is Gaussian and is spread evenly across the digitized spectrum. The video filter attenuates the out of band noise by 20 dB. Quadrature demodulation requires the removal of the negative frequency passband image either before or after the complex mixer. Since decimation aliases that image onto the desired signal, the phase splitter has to attenuate the image by at least 85 dB.

The complex mixer shifts the echo spectrum down by ω_c so the positive frequency component becomes a complex baseband signal. The phase splitter suppresses the negative frequency part which is shifted to $-2\omega_c$. The resulting complex baseband signal has a 2 MHz bandwidth, so we can decimate it 4:1 without aliasing to minimize the processing load in the matched filter.

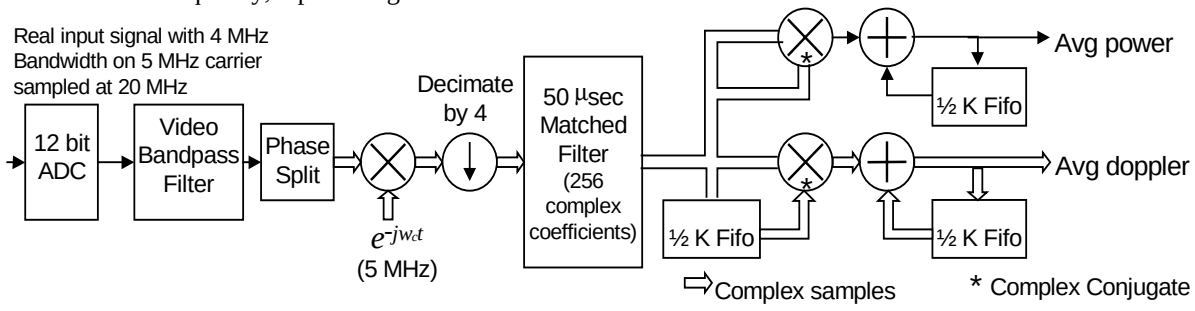


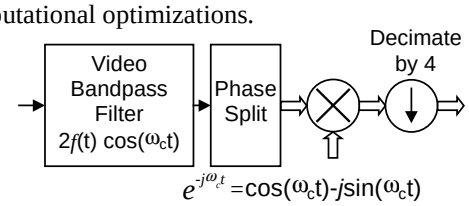
Figure 1. Radar signal processor (one of four channels shown)

3. Making it happen in an FPGA

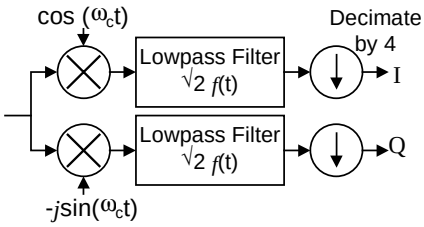
The matched filter in each channel alone requires over 1000 multiplies per sample, a processing load that easily overwhelms a microprocessor. By taking advantage of the parallelism available in an FPGA and using a few algorithmic tricks, we can easily handle the processing for two channels in one FPGA.

3.1. Algorithmic efficiencies

The video filter and the phase splitter (a Hilbert transform filter) can be combined into a single analytic filter, which is still a bandpass filter, but it only passes the positive frequency components of the real input signal. The impulse response of that filter has to be complex, so the filter requires two real filters for implementation. That composite filter response can be demodulated to move it to the other side of the mixer[4] as shown in Figure 2. Note that the translated filter response is a low pass filter. Rearranging the demodulator provides an opportunity for significant computational optimizations.



a) conventional arrangement



b) Filter translated to follow mixer

Figure 2. Equivalent demodulator structures.

Decimating the filter output involves discarding three of every four output samples. Since we do not use the discarded samples, there is no need to perform the associated computations [5]. Figure 3 shows the computations for three successive retained outputs for a FIR filter when the output is decimated by four. Inspection reveals that each coefficient only multiplies every fourth sample. The filter can be decomposed into a sum of 4 sub-filters, each of which filters a particular "phase" of the input signal. One filter phase is highlighted in the illustration. Each sub-filter operates at the decimated sample rate rather than the input rate, so even though there are as many taps as the original filter, the processing load is a quarter of what it was. Figure 4 shows the resulting decimate by four filter structure.

$$
\begin{aligned}
Y_{k+0} &= \boxed{x_k C_0} + x_{k-1}C_1 + x_{k-2}C_2 + x_{k-3}C_3 + \boxed{x_{k-4}C_4} + \ldots \\
Y_{k+4} &= x_{k+4}C_0 + x_{k+3}C_1 + x_{k+2}C_2 + x_{k+1}C_3 + x_k C_4 + \ldots \\
Y_{k+8} &= x_{k+8}C_0 + x_{k+7}C_1 + x_{k+6}C_2 + x_{k+5}C_3 + x_{k+4}C_4 + \ldots
\end{aligned}
$$

Figure 3. Filter output after decimating by 4

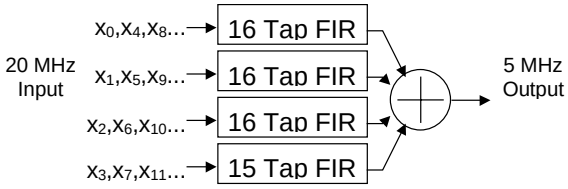


Figure 4. 63 tap decimate by 4 FIR filter.

The *N* coefficients of a FIR filter with linear phase are symmetric so that $C_i = C_{N-i}$. The number of multiplies can be cut in half by folding the tapped delay line and pre-adding sample pairs before multiplying by the coefficient as is shown in Figure 5.

$$Y_k = x_k C_0 + x_{k-1}C_1 + x_{k-2}C_2 + x_{k-3}C_3 + \ldots + x_{k-n}C_n$$

$$Y_k = (x_k + x_{k-n})C_0 + (x_{k-1} + x_{k-n+1})C_1 + \ldots + (x_{k-n/2})C_{n/2}$$

Figure 5. Symmetry reduces multiplications

As with many systems, we have some freedom in the selection our input sample rate. If it is set to be exactly 4 times the local oscillator, then the complex local oscillator, e^{-jwt}, is represented by the repeating complex sequence: 1, -j, -1, j. When followed by a decimate by four filter, the LO value is constant for each phase of the filter. The mixer can be eliminated by weighting each of the sub-filters with the LO value corresponding to that phase [6]. Note that the phase angles selected weight half of the sub-filters with zero, eliminating those sub-filters as shown in Figure 6.

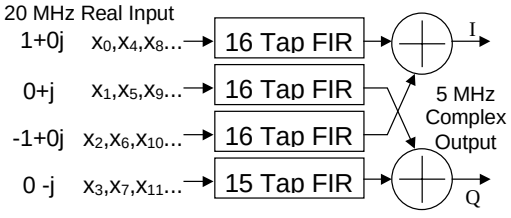


Figure 6. Complete decimating downconverter with 63 tap video filter

If the length of the filter is odd and its coefficients are symmetric, the I and Q filters each use only half of the 32 unique coefficients (I uses 16, Q uses the other 15). Using the symmetry cuts the multiplications in the filter in half again. We took advantage of decimation and symmetry in our demodulator. The matched filter is non-symmetric so these tricks do not apply there.

3.2. Bit serial arithmetic

Modern FPGAs can easily handle synchronous designs with clocks faster than 100 MHz with extensive pipelining and careful design. In designs like this, where the data rate is a comparatively low 5 MHz, a conventional bit parallel design leaves much of the FPGA's capability unused.

Bit parallel designs process all of the bits of a data word simultaneously. In contrast, a bit serial structure processes the data one bit at a time. The advantage is that all of the bits pass through the same logic, resulting in a huge reduction in the required hardware. Typically, a bit serial design requires only about $1/n^{th}$ of the hardware needed for the equivalent *n*-bit parallel design[7][8]. The price of this logic reduction is that the serial hardware takes *n* clock cycles to execute, while the equivalent parallel structure executes in one.

3.3. Distributed arithmetic

Bit serial arithmetic alone doesn't reduce the size of the complex matched filter enough to fit it in an FPGA (A thousand 12 bit serial multipliers alone would just about fill a Virtex XCV1000. Another trick, distributed arithmetic [9][10], is the key to compacting the filters to a manageable amount of hardware. This technique rearranges the multiplies and adds of a sum of products at the bit level to take advantage of small tables of precomputed sums.

The distributed arithmetic approach is an inherently serial process, but the computations for each bit can be performed in parallel to obtain more performance if necessary. Figure 7 shows a four tap FIR filter made of serial by parallel "scaling accumulator" multipliers. The AND gates compute a 1 x *n* partial product which is added to the shifted sum of the previous partial products for each bit in the serial input. The coefficients, C_i, are generally constants. The first shift register serializes the input and the remaining ones delay the input by one sample time each.

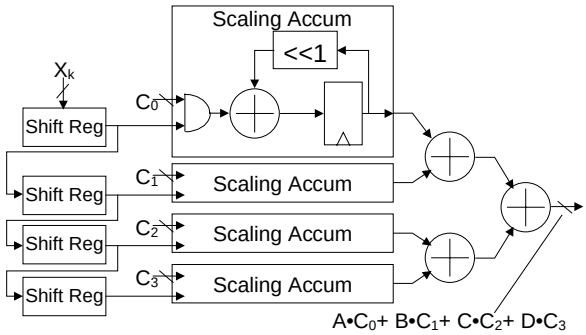


Figure 7. Four tap serial FIR filter

Each of the scaling accumulators simply sums the 1 x *n* partial products for the multiplication of the associated coefficient. The adders can be rearranged so that the 1 x *n* partial products from all the coefficients are summed before adding the result to the previous bit sum as shown in Figure 8. Note that the 1x *n* multiplies are still done at the beginning; all we have done is changed the order that we sum the partial products. This simple change directly eliminates *k*-1 registered adders in a *k* tap serial filter.

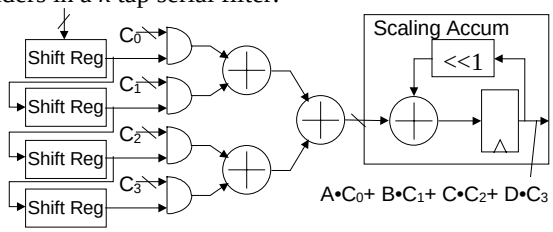


Figure 8. Serial FIR filter rearranged to adder tree and one scaling accumulator

Since the coefficients are constants, the AND gates and adder tree can be reduced to a look-up table with a four bit input. The sixteen table entries are the sums of the 1 x *n* products of the inputs and coefficients for all the possible input combinations, as shown in Figure 9. The table is made wide enough to accommodate the largest sum without overflow.

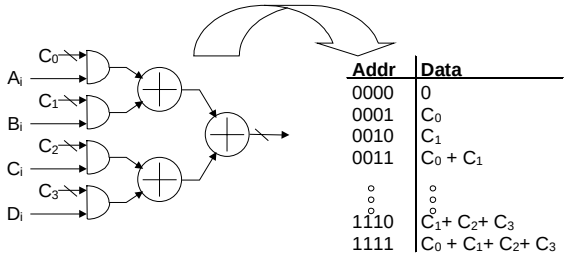


Figure 9. Serial adder tree and coefficients can be implemented as look-up table in ROM

The size of the look-up table grows exponentially with the number of inputs. Rather than creating a larger table, we combine the outputs of smaller tables with adders. That way, the size of the circuit grows linearly with the number taps instead of exponentially. This architecture is well suited to FPGAs that use 16x1 look up tables as their basic logic resource.

The actual circuit is heavily pipelined to increase the circuit throughput at the expense of clock latency. A serial distributed arithmetic filter with 12 bit coefficients implemented in the slowest speed grade (XCV1000-4) Xilinx Virtex is capable processing at more than 150 Mbps, regardless of the number of taps. A 256 tap filter with 12 bit coefficients occupies about 715 Virtex configurable logic blocks (CLBs), which is less than 3 CLBs per tap.

3.4. Interleaved processing

The filter design can be clocked at more than twice the bit rate we require for 12 bit inputs, even in the slowest parts. We can take advantage of the excess capacity by multiplexing two channels through one set of hardware when the processing for both channels is identical. A distributed arithmetic filter is modified to handle two channels by simply doubling the delay between taps and adding a clock delay in the accumulator feedback loop. The delay in the accumulator loop lets us accumulate two sums simultaneously: one sum is accumulated on the odd clocks, and one on the even clocks. The input to the filter is interleaved serial data so that one channel's data is shifted in on even clocks and the other on odd clocks. Both channels must use the same filter coefficients.

The demodulator circuit uses the same filter coefficients for all the channels, so by interleaving bits, one demodulator circuit can handle two channels. The matched filter coefficients are unique to each channel so it cannot be shared among channels. However, since the matched filter has complex inputs and coefficients, it processes the I and Q data streams with identical pairs of real filters. We can interleave the 'I' and 'Q' samples so that we implement the complex matched filter with a pair of two channel filters instead of four filters as shown in Figure 10.

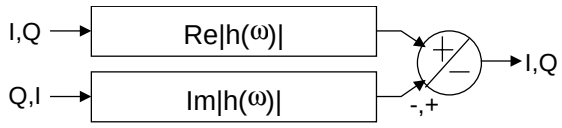


Figure 10. Complex filter using interleaved sample 2 channel filters

4. The FPGA solution

A two channel processor fits into one Xilinx Virtex XCV1000-4 FPGA using the optimizations described. The two channel processor consists of a 2 channel decimating demodulator, (Figure 4 modified for 2 interleaved channels), a pair of complex matched filters (Figure 10) and a pulse pair processor for each channel. The decimator and matched filter both have 12 bit coefficients and 13 bit inputs (we added a 13th bit to take advantage of the SNR improvement out of the video filter). The demodulator and complex matched filters for two channels occupy approximately 3100 CLBs - about half of the FPGA's logic cells.

The complex multipliers in the doppler pulse pair process are implemented with 24 x 26 scaling accumulator multipliers, (the 24 bit parallel input is the previous pulse sample). The FIFO buffers for the previous pulse and averaging are implemented in the Virtex Block RAM, so even the memory for the processor is inside the FPGA.

The entire 2 channel design occupies approximately 55% of the logic plus all of the block RAM in the XCV1000. The design is implemented on an Annapolis Microsystems Wildstar^{TM} board which contains 3 Virtex XCV1000-4s. We've used two of the FPGAs for the four channel processor, leaving the third for future processing needs. The design is clocked at 130 MHz to allow two interleaved 13 bit 5MHz processes. Timing analysis indicates the design is capable of better than 150MHz, a healthy 15% margin.

5. Conclusions

In this paper we have discussed the implementation of a doppler weather radar processor in an FPGA. We presented a series of design optimizations and hardware tricks that allowed us to perform the equivalent of over 10 billion multiplies per second in one part. The tricks and techniques discussed here are extendable to a large class of signal processing applications in communications, imaging and other areas, all of which can benefit from an FPGA implementation.

6. References

[1] Serafin, R.J., "Meteorological Radar," in Skolnik, M., *Radar Handbook, 2nd Edition*, McGraw-Hill, 1990.

[2] Weil, T.A., "Transmitters," in Skolnik, M., *Radar Handbook, 2nd Edition*, McGraw-Hill, 1990.

[3] Farnett, E.C. and Stevens, G.H., "Pulse Compression Radar," in Skolnik, M., *Radar Handbook, 2nd Edition*, McGraw-Hill, 1990.

[4] Lee, E.A. and Messerschmitt, D.G., *Digital Communication, 2nd Edition*, pp. 203-206, Kluwer Academic Publishers, Norwell, Massachusetts, 1994.

[5] Vaidyanathan, P.P., *Multirate Systems and Filter Banks*, Chapter 4, Prentice-Hall, Englewood Cliffs, NJ., 1993.

[6] Frerking, M.E., *Digital Signal Processing in Communication Systems*, Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[7] Denyer, P. and Renshaw, D.,*VLSI Signal Processing: a Bit Serial Approach*, Addison-Wesley Publishers LTD, England, 1985.

[8] Andraka, R.J., "FIR Filter Fits in an FPGA Using a Bit Serial Approach," *Proceedings, 3rd Annual PLD conference and exhibit*, March 30-31, 1993. CMP Publications, Manhasset, NY, 1993.

[9] Peled, A. and Liu, B., "A New Hardware Realization of Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.

[10] Goslin, G and Newgard, B., "16 Tap, 8 Bit FIR Filter Applications Gude," Xilinx Application Note, Nov, 1994. Available: http://www.xilinx.com/appnotes/fir_filt.pdf