VLSI Chip Architecture for Real-Time Ambiguity Function Computation

Nenad Marinovich
Department of Electrical Engineering
City College of CUNY
New York, NY 10031

Vojin G. Oklobdzija
Department of Electrical Engineering and
Computer Science
University of California
Davis, CA 95616

Abstract

A chip architecture is developed that evaluates the cross-ambiguity function samples on an arbitrary sampling grid at real-time radar rates. It can be scaled to match different date lengths and sampling grid sizes. It is shown to be potentially quite useful for a wide range of typical radar system.

1. Introduction

It is well known that, under ideal conditions, the optimal procedure for joint estimation of the delay and Doppler shift of a radar echo requires computation of the complex cross-ambiguity function between the transmitted and the received signals. Even under non-ideal conditions, in a real world operating environment, this procedure is considered to be the best suboptimal one. However, prohibitively high computational complexity and the lack of efficient algorithms and appropriate hardware have been preventing its implementation in practice. Recently, a new algorithm for efficient computation of the decimated cross-ambiguity function on arbitrary sampling grids have been developped by Auslander, Gertner, and Tolimieri (AGT algorithm)[1,2]. Its important feature is that the computations on complementary, decimated sampling grids can be performed independently, in parallel, in the same time it takes to compute the FFT of the received data vector. Combined with the progress in VLSI technology, this makes the implementation of the cross-ambiguity function based receiver feasible at present time. In this paper, based on a simple modification of the AGT algorithm, we develop a VLSI chip architecture for real-time computation of the cross-ambiguity function magnitude. Using actual radar examples, we discuss the design trade-offs, and demonstrate the scalability of the architecture needed to accommodate different data lengths and sampling grid sizes.

2.Background

Let x(t) and y(t) denote complex envelopes of the transmitted and received signals, respectively, and x(n) and y(n) sequences of their samples. The complex cross-ambiguity function between the received and transmitted signals is defined as:

$$(1) \qquad A_{yx}(\nu,\tau)=\int_{-\infty}^{\infty} y(t)x^{*}(t-\tau)e^{j2\pi\nu t}dt$$

Regularly sampled CAF is defined on an L x L sampling grid as

$$(2) \qquad A_{yx}(k\Delta\nu,l\Delta\tau)=\sum_{i=0}^{L} y(i)x^{*}(i-l)e^{j\frac{2\pi}{L}ik}dl$$

$$0\le l<L \qquad -\frac{L}{2}\le k<\frac{L}{2}$$

where L>C+D-1, with C and D being the transmited and received sequence lengths, respectively. Given that data volume L is very large, typically in thousands, the high computational cost in Eq. (2) becomes readily apparent even when FFT is used. In addition, what makes Eq. (2) impractical to use is the requirement, typically encountered in practice, that the samples of the CAF need to be computed on a considerably smaller sampling grid corresponding to a desired quantization of delay and Doppler shifts. This is indicated in Fig. 1 where P and R are the numbers of Doppler and range measurement bins, respectively. However, the efficient computation of the CAF samples on a desired grid is possible with the help of the new AGT algorithm [1,2] that is outlined below.

Initial step requires the computation of the discrete Zak transform (DZT)[1] of the received data vector. If L=MN, the DZT is defined as an M x N matrix

$$(3) \qquad Z_{y}(r,s)=\sum_{i=0}^{N-1} y(s+iM)e^{-j\frac{2\pi}{N}ir}$$

This computation can be visualized in the following way: first, rearange the data vector into an M x N matrix by filling the rows with successive blocks of M data samples; then, replace each column by its discrete Fourier transform (DFT) In the next step, pointwise multiplication of the received data DZT with the precomputed DZT of the transmitted data is performed, followed by the 2-D inverse DFT of the result. This gives

1058-6393/91 $01.00 © 1991 IEEE

the L samples of the CAF on a decimated M x N sampling grid, with sample spacing along the Doppler and delay axes being N $\Delta\nu$ and M $\Delta\tau$ respectively:

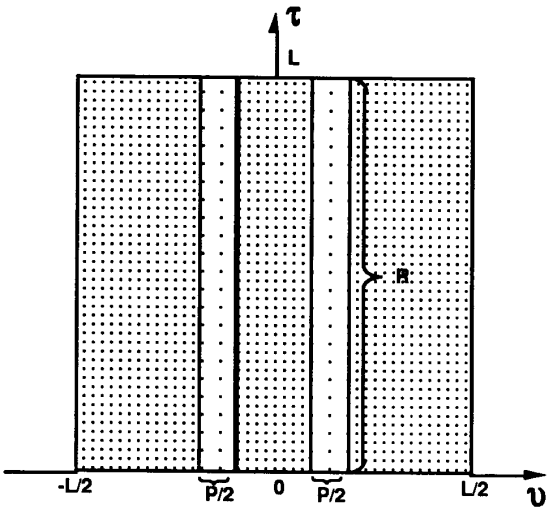

Fig. 1 Regular (fine) and desired (coarse) sampling grids

$$(4)\quad A_{yx}(mN\Delta\nu, nM\Delta\tau) = 2D - DFT^{-1}\{Z_y(r,s)Z^*$$

$$= \frac{1}{MN}\sum_{r=0}^{N-1}\sum_{s=0}^{M-1} Z_y(r,s)Z^*{}_x(r,s)e^{j2\pi(\frac{rn}{N}+\frac{ms}{M})}$$

When FFT algorithms are used for DFT computation, this procedure takes about the same amount of computation as an L-point FFT of the raw data. An important property of the CAF that permits its computation on arbitrary domains [2] is the time and frequency shift property:

$$(5)\quad x_{\phi\theta}(t) = x(t-\theta)e^{-j2\pi\phi t} \Rightarrow$$

$$A_{yx_{\phi\theta}}(\nu,\tau) = e^{j2\pi\nu\theta}A_{yx}(\nu+\phi,\tau+\theta) \Rightarrow$$

$$|A_{yx}(\nu+\phi,\tau+\theta)| = |A_{yx_{\phi\theta}}(\nu,\tau)|$$

In particular, to compute the magnitude of the CAF on another sampling grid with arbitrary offsets ϕ and θ, we use the AGT algorithm with precomputed DZT for those offsets:

$$(6)\quad |A_{yx}(mN\Delta\nu+\phi, nM\Delta\tau+\theta)| =$$

$$|A_{yx_{\phi\theta}}(mN\Delta\nu, nM\Delta\tau)|$$

For each desired offset pair (ϕ, θ), we repeat the computation of (6) using the AGT algorithm, Eqs. (3) and (4), to obtain the samples of the CAF magnitude on the desired uniform sampling grid. Note that all the computations performed for different ϕ , θ pairs after the DZT of the received data is determined, are independent and can be performed in parallel.

3.Algorithm

The computation outlined above is too wasteful for the non-uniform sampling grids such as the one displayed in Fig. 1. However, a simple modification permits its use in such a case, too. Namely, instead of computing all M x N values in (6) for each offset pair, we should only compute N samples along the m=0 axis

$$(7)\quad |A_{yx}(0+\phi, nM\Delta\tau+\theta)| =$$

$$= |2D - DFT^{-1}\{Z_y(r,s)Z^*{}_{x_{\phi\theta}}(r,s)\}|_{m=0}$$

$$= |\frac{1}{MN}\sum_{r=0}^{N-1} e^{j\frac{2\pi}{N}rn}\sum_{s=0}^{M-1} Z_y(r,s)Z^*{}_{x_{\phi\theta}}(r,s)|$$

for each pair of offsets

$$(8)\quad \phi = \pm(\nu_0 + p\alpha) \qquad p = 0,1,\ldots,\frac{p}{2}-1$$

,

$$\theta = q\beta \quad \beta = \frac{M\Delta\tau}{Q} \quad q = 0,1,\ldots,Q-1 \quad Q = \frac{R}{N}$$

Note that the 2-D M x N inverse FFT in (4) has been replaced by an N-point inverse FFT and N x (M-1) additions. The total number of multiplications required if radix-2 Cooley-Tukey FFT is used is

(9) PQN(log N + 1)/2 + MN(log N)/2 =
=(PR + L)(log N)/2

Our goal is to develop a modular architecture that can be integrated on a chip while allowing the independent computations to proceed in parallel. With that in mind we organize the computation (7) in the following reduced grid CAF algorithm (RGCAF) that minimizes necessary RAM storage:

BEGIN RGCAF
  FOR *each column of the input data matrix* DO
    - *compute its* FFT
    FOR *each* ϕ, θ *pair* DO IN PARALLEL
      - *pointwise multiply with respective column of the precomputed reference* DZT *of* x ϕ,θ
      - *accumulate the resulting column vector over all input columns*
    DONE
  DONE
  FOR *each* ϕ,θ *pair* DO IN PARALLEL
    - *compute the inverse* FFT *of the result*
    - *output the magnitude of the result*
  DONE
END RGCAF

Visual illustration of this computation is given in Fig. 2.

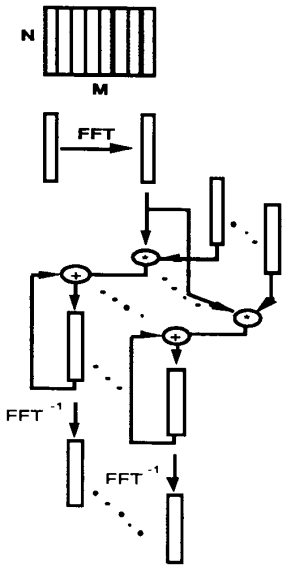

Fig.2 Computation Flow

4.Architecture

Examination of the RGCAF algorithm indicates that two processing modules are required: one to compute the N-point FFT of the input data matrix column - input module (IPM), and the other to perform the pointwise vector multiplication/accumulation and the inverse N-point FFT - parallel processing module (PPM). Data-path that computes the RGCAF algorithm is shown in Fig 3. The core computation in both modules is that of an N-point FFT. Least complex hardware implementation of the FFT for our purposes is based on repetitive use of a basic radix-2 Cooley-Tukey FFT butterfly processing element (PE) and requires two N-word RAM buffers to alternate the storage of input and output vectors during successive stages of the radix-2 Cooley-Tukey FFT algorithm. In addition, an N-word ROM of FFT coeficients is required in each processing module, and different PPM's have an L-word ROM to store the precomputed DZT of the delay and Doppler shifted transmitted signal for different ϕ,θ shifts. Finally, a magnitude estimator is a combinational network that generates an approximation of a complex number magnitude according to the following simple rule:

$$A = R + jI \tag{10}$$

$$|A| = \max\left\{ \begin{matrix} |R| \\ \frac{7}{8}|R| + \frac{1}{2}|I| \\ \frac{1}{2}|R| + \frac{7}{8}|I| \\ |I| \end{matrix} \right\}$$

This magnitude estimator has maximum error of 3% [3].

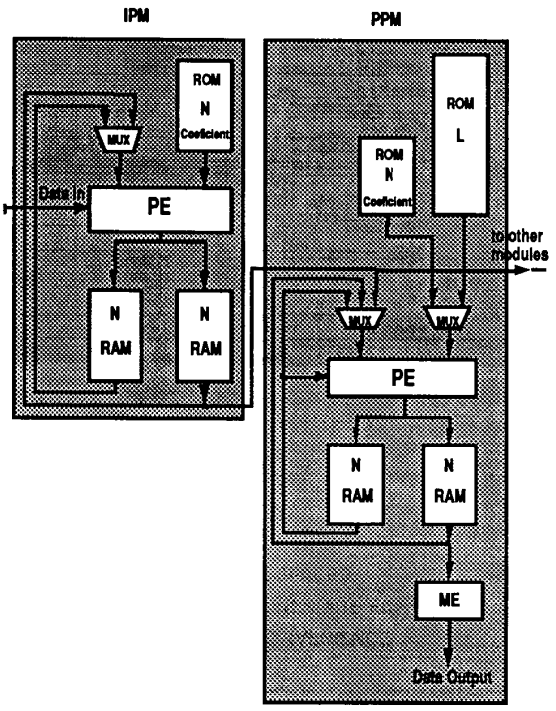

Fig. 3. Organization of the Data-Path

To accommodate the large dynamic range typical in radar applications and the fast growth of the signal through the processing chain, the complex data is internally represented using a following hybrid floating-point format[3]:

$$(m_r + jm_i)2^e \tag{11}$$

where m_r and m_i are real and imaginary component mantissas, represented as 2's complement fractions, and e is the positive exponent whose initial value is set to zero for the input signal. Since the data amplitude will only grow during the processing, exponent is limited to be a positive integer. Rather broad range of radar applications can be accommodated with 11 bit mantissa and 5 bit exponent which requires a 27-bit complex word size[3]. The floating-point butterfly processing element which may pass through the data from its left side input, and permits pointwise vector multiply/accumulate operation is shown in Fig. 4.

It is implemented as a four-stage pipeline that computes the inverse FFT in the follwing manner:

1) - first butterfly input is loaded from the input RAM buffer into SR1;
2) - SR2 is loaded with the product of SR1 and 1/2 of an FFT coefficient;

- SR3 is loaded from the input RAM buffer with 1/2 of the second butterfly operand;
3) - difference SR3-SR2 is placed in SR5;
- sum SR3+SR2 is placed in SR4;
- SR1 is loaded from the input RAM buffer with the third input operand;
4) - first butterfly output from SR5 is stored into the other RAM buffer;
- SR5 is loaded with SR4;
- SR2 is loaded with the product of SR1 and 1/2 of an FFT coefficient;
- SR3 is loaded from the input RAM buffer with 1/2 of the fourth input operand;
5) - second butterfly output from SR5 is stored into the other RAM buffer;
- difference SR3-SR2 is placed in SR5;
- sum SR3+SR2 is placed in SR4;
- SR1 is loaded from the input RAM buffer with the fifth input operand;
6) - third butterfly output from SR5 is stored into the other RAM buffer;
- SR5 is loaded with SR4;
- SR2 is loaded with the product of SR1 and 1/2 of an FFT coefficient;
- SR3 is loaded from the input RAM buffer with 1/2 of the sixth input operand;
7) - fourth butterfly output from SR5 is stored into the other RAM buffer;
- difference SR3-SR2 is placed in SR5;
- sum SR3+SR2 is placed in SR4;
- SR1 is loaded from the input RAM buffer with the seventh input operand;
.....continue.....

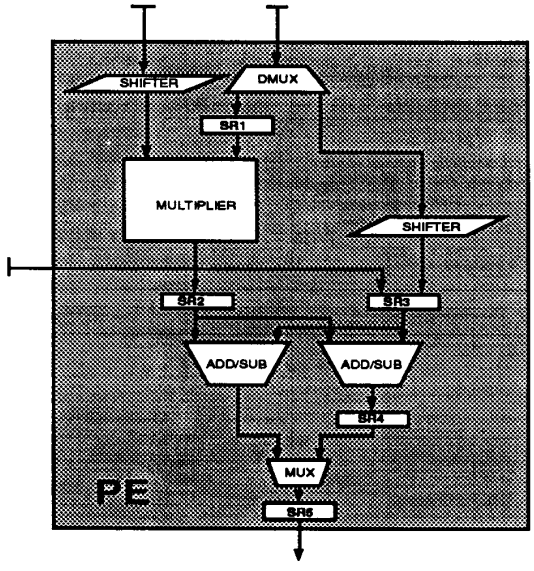

Fig.4. Organization of the Processing Element

This process is continued until all the data from the input RAM buffer has been processed. Then the role of the input and output buffer is exchanged and the whole process is repeated for the next stage of the FFT computation. In computation of the forward FFT, division of operands by 2 is not performed.
The whole FFT computation takes approximately N(log N) clock cycles (neglecting the pipeline latency). We can now estimate the total execution time to be approximately equal to

(12) $Tx=[L(\log N + 1)]Tc$

This is less than the time needed for the computation of a single L-point FFT of the raw data with circuits of comparable complexity. Regularity and modularity of the proposed architecture make it easy to integrate. It can be scaled to accomodate different data lengths L. On one extreme, for huge data lengths, an IPM and a single PPM would be put on a single chip and P x R/N chips will be required to generate P x R CAF samples. On the other, for short data lengths, an IPM and many PPM's would fit on a single chip. How many chips would be required depends on the desired number of Doppler bins P, and on how many of the R range bins are computed per PPM. The trade-offs involved are discussed in the next section.

5.Radar Examples

Practical implications of the proposed chip architecture can be assessed by considering how could it be used in actual radar systems. With this goal, we consider several typical pulsed Doppler radar systems [4]. We estimate the equivalent gate count of the required processing modules in each case, and estimate the chip count knowing that, with the current state of the art, up to about 500,000 gates are available on a single chip. With the present state of the art, it is estimated that the clock cycle of 10nsec is sufficient for proper operation of the processing element in Fig. 4 and for memory access[5]. This value is used when estimating the execution times.

A. Airborne early warning radar (AEWR)

This radar operates in the UHF band, with the range 300 nautical miles = 1.8msec, the pulse repetition frequency PRF=300Hz, data block L=32K samples, R=4K range bins, and P=8 Doppler bins. Using N=1K and M=32, we estimate the gate count to be 120K for IPM and 220K for PPM.

Consequently, only one PPM together with an IPM can be put on single chip that will compute 1K range samples. Therefore, four chips are required per Doppler bin, and total of 4P=32 chips are required.

Total execution time is approximately T=3.2μsec. The input data block rate is determined by the longer of the maximum time-of-flight (2 x 1.8msec) and the pulse repetition interval (3.3msec). It equals one L-point data block per 3.6 msec in this case. Therefore, to keep up with the incoming data rate, one 32-chip set is sufficient.

B. 2-D Air surveillance radar (ASR)

This is an S-band radar, with range 60 nmi = 0.36msec, PRF=1.2KHz, data block L=4K, R=1K range bins, and P=8 Doppler bins. Using N=1K and M=4, the estimated gate count is 20K for IPM and 125K for PPM.

With room to spare, three PPM's can fit on a single chip together with an IPM and all the range samples for three Doppler bins can be computed with a single chip. Three chips are required for all the desired Doppler bins.

Each CAF computation can be completed in approximately 0.44msec. The input data block rate is one block per 0.83msec. Again, one set of three chips is sufficient to keep up with the incoming data.

C. Portable battlefield reconnaissance radar (BRR)

This is a Ku-band radar, with range 6 nmi = 36μsec, PRF=12KHz, data block L=1K, R=256 range bins, and P=10 Doppler bins. Using N=256 and M=4, the estimated gate count is 6K for IPM and 32K for PPM.

Twelve PPM's can fit on a single chip which is sufficient for the computation on all the range and Doppler bins.

Execution time is approximately 92μsec, which can keep up with the data coming in at the rate of one block per 100μsec. In this case one chip is sufficient to implement the radar's Doppler processor.

D. Airborne intercept radar (AIR)

Operating in X-band this nose-mounted radar has range of 30nmi = 180usec, PRF=150KHz, data block L=1024, R=256 range bins, and P=50 Doppler bins. Using N=256 and M=4, the estimated gate count is 6K for IPM and 32K for PPM.

Fifteen PPM's can fit on a single chip and it can compute all the range samples for fifteen Doppler bins. Four chips are needed for all the required bin computations.

It should take approximately 92μsec to complete the computation. To keep up with the incoming data rate of one block per 360μsec only one set of four chips is needed.

6.Summary

A VLSI chip architecture for real-time cross-ambiguity function computation in radar systems was developed and its practical implications were assessed. It is based on a modification of a recent algorithm for decimated ambiguity function computation[1,2].

Bulk of the chip is memory, both RAM and ROM. FFT is performed by repetitive use of a single butterfly computing element. With the present state of the art, the resulting low circuit complexity and simple control allow integration of the processing modules on one or more chips, depending on the radar system requirements, avoiding the chip-crossing penalties and improving speed.

The architecture is regular, easy to integrate, and modular. It can be scaled to match different data lengths. To accomodate various sampling grid sizes and input data rates, chips can be parallelized and pipelined with virtualy no overhead. Potential utility of this architecture for a broad range of radar applications was illustrated on several examples of typical systems.

7.References

[1] L. Auslander, I.G. Gertner, and R. Tolimieri, "*The discrete Zak transform application to time-frequency analysis and synthesis of nonstationary signals*", IEEE Trans. Sig. Proc., vol 39, no. 4, pp. 825-835, April 1991.

[2] L. Auslander and I.G. Gertner, "*Computing the ambiguity function on various domains*", submitted for publication.

[3] J.H. McClellan and R.J. Purdy, "*Applications of digital signal processing to radar*", Chapter 5 in "*Applications of Digital Signal Processing*", A.V. Oppenheim (ed.), Prentice-Hall, 1978.

[4] D.C. Schleher, "*MTI and Pulsed Doppler Radar*", Artech House, 1991.

[5] R.K. Montoye, E. Hokenek, and S.L. Runyon, "*Design of the IBM RISC System/6000 floating-point unit*", IBM Jour. Res. and Devel., vol 34, no. 1, pp. 59-70, Jan. 1991.