Reconfigurable Computing for Space-Time Adaptive Processing

Nikhil D. Gupta, *John K. Antonio*, and *Jack M. West*
Department of Computer Science
Box 43104
Texas Tech University
Lubbock, TX 79409-3104 USA
Tel: 806-742-1659
{gupta, antonio, west}@ttu.edu

1. Introduction

Space-time adaptive processing (STAP) refers to a class of signal processing techniques used to process returns of an antenna array radar system [4]. STAP algorithms are designed to extract desired target signals from returns comprised of Doppler shifts, ground clutter, and jamming interference. STAP simultaneously and adaptively combines the signals received on multiple elements of an antenna array – the spatial domain – and from multiple pulse repetition periods – the temporal domain.

The output of STAP is a weighted sum of multiple returns, where the weights for each return in the sum are calculated adaptively and in real-time. The most computationally intensive portion of most STAP approaches is the calculation of the adaptive weight values. Calculation of the weights involves solving a set of linear equations based on an estimate of the covariance matrix associated with the radar return data.

Existing approaches for STAP typically rely on the use of multiple digital signal processors (DSPs) or general-purpose processors (GPPs) to calculate the adaptive weights. These approaches are often based on solving multiple sets of linear equations and require the calculation of numerous vector inner products. This paper proposes the use of FPGAs as vector co-processors capable of performing inner product calculation.

Two different "inner-product co-processor" designs are introduced for use with the host DSP or GPP. The first has a multiply-and-accumulate structure, and the second uses a reduction-style tree structure having two multipliers and an adder.

2. STAP Weight Calculation

2.1 Basic Formulation

The STAP algorithm assumed here is known as K^{th}-order Doppler factored STAP, which is classified as a partially adaptive technique. Due to the space limitation, it will not be possible to fully explain this algorithm. Instead, the focus here will be on the necessary notation and core calculations required to determine the values of the adaptive weights. For more information on STAP, the reader is referred to [1, 4].

Determining the values for the *n*-vector of adaptive weights, denoted by $\vec{w}$, involves solving a system of linear equations of the form:

$$\Psi \vec{w} = \vec{s}\ , \tag{1}$$

This work was supported by DARPA under contract number F30602-97-2-0297

where $\vec{s}$ is a known *n*-vector called the steering vector and Ψ is an estimate of the covariance matrix, which is determined based on the sampled radar returns. Ψ is derived based on space-time data matrix *X*, which is an n ´ N matrix defined by: $X = [\vec{x}_1\ \vec{x}_2\ \ldots \vec{x}_N]$. Based on this the definition, Ψ is given by:

$$\Psi = \frac{1}{N} X\ X^H \tag{2}$$

2.2 QR-Decomposition and Conjugate Gradient

The QR-decomposition approach is a direct approach for solving a system of linear equations. The QR approach always gives an exact solution and the complexity of the algorithm is fixed. It involves performing a QR-decomposition on the matrix X^T, the result of which is an $N \times N$ orthogonal matrix *Q* and an $n \times N$ upper triangular matrix *R* such that *X* = *QR.* The final result is obtained by forward and backward substitution. For more details the reader is referred to [1].

The conjugate gradient approach is an iterative method that provides a general means for solving a system of linear equations [2]. For the system of equations given in Eq. (1), it is based on the idea of minimizing the following function:

$$f(\vec{w}) = \frac{1}{2}\vec{w}^T \Psi \vec{w} - s\vec{w}\ . \tag{3}$$

The function *f* is minimized when its gradient is zero, i.e., $\nabla f = \Psi\vec{w} - \vec{s} = 0$, which corresponds to the solution to the original system of linear equations. The very repetitive and regular numerical structure of the conjugate gradient update equations makes it a prime candidate for implementation on an FPGA system.

Numerical studies were conducted using Matlab implementations of the QR-decomposition and CG methods on actual STAP data collected by the Multi-Channel Airborne Radar Measurement (MCARM) system of Rome Lab [3]. Further details of this study can be found in [6].

3. Inner-Product FPGA Co-Processor

Each of the two methods outlined above requires calculating a number of inner products. Given enough resources, all the inner products could be done in parallel on FPGAs. Because the available system has only two FPGAs [5], the computations was divided among the host processor and the FPGA board. The two schemes that were implemented are outlined below. For both schemes, the data vectors are assumed to be in block-floating-point format [9]. Additionally, the multiplier implementation is based on discussion in [7] and the adder unit uses 4-bit carry-look-ahead adders [8] in each stage of the adder pipe.

3.1 Multiply-and-Accumulate Implementation

In the first implementation shown in Figure 1, the FPGA is configured to perform the multiply-and-accumulate operations on the input vectors. The implementation consists of a multiply unit and an accumulator, which is composed of a normalization unit and an adder. The normalization unit shifts the binary point of the mantissa and makes a compensating adjustment to the exponent prior to the addition. The output of the adder is fed back and accumulated with the next product term.

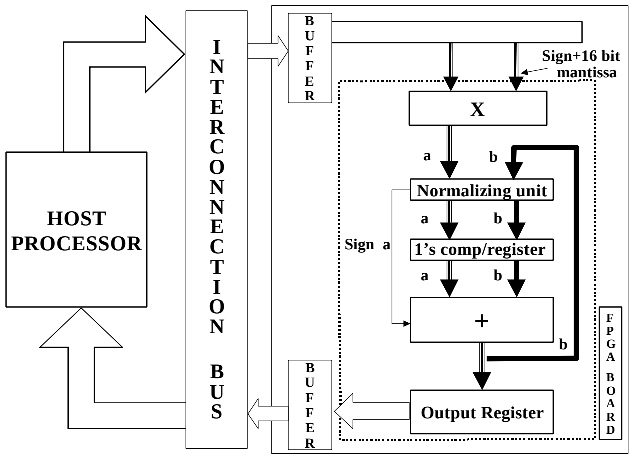

Figure 1: Block diagram implementation of the multiply-and-accumulate unit on WildOne FPGA board.

The single cycle multiply-and-accumulate is achieved by pipelining each unit of the implementation. This unit reads in two operands and performs two operations per cycle. Thus, the unit reduces two *N*-vectors to a constant number of partial sums equal to the number of stages in the accumulator pipe. The implementation allocates approximately 88% of the configurable logic blocks (CLBs) on the Xilinx 4028EX FPGA. The implementation can be clocked at 40MHz, thus giving a throughput of 80 million block-floating-point operations per second.

3.2 Multiply-and-Add Implementation

Figure 2 illustrates the second implementation that performs an inner product, i.e., a multiply-and-add operation on the two input vectors. The design incorporates two 16-bit multiply units and an adder. By using this approach, two multiplies can be performed in parallel, and afterwards, the adder computes the sum of the two products.

A challenge associated with this implementation is that four 16-bit input operands, i.e., 64 bits, are required per computation cycle. Unfortunately, the data-path to the FPGA board is only 36-bits wide. The solution to this problem involves clocking the input state machine at twice the frequency of the multiply-and-add state machine, and registering the first two operands for one input state machine clock cycle.

The multiply-and-add unit reads in four operands and performs three block-floating-point operations per cycle. Thus,

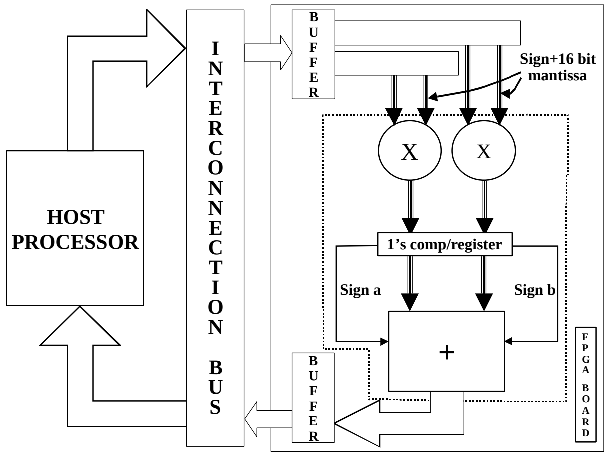

Figure 2: Block diagram implementation of the multiply-and-add unit on WildOne FPGA board.

the two input *N*-vectors are reduced to an *N*/2-vector of partial sums. This implementation, however, involves an additional *N*/2 addition operations to obtain the inner product result. For this implementation, approximately 99% of the available CLBs on the Xilinx 4028EX FPGA are required. In summary, for a fixed clock rate, the second design can provide a higher throughput, but requires more computation from the host (to perform the final summation of the partial sums).

4. References

[1] K. C. Cain, J. A. Torres, and R. T. Williams, "*Real-Time Space-Time Adaptive Processing Benchmark*", Mitre TR: MTR 96B0000021, Mitre, Bedford, MA, February 1997.

[2] D. G. Luenberger, *Linear and Nonlinear Programming*, Second Edition, Addison-Wesley, Reading, MA, 1984.

[3] Real-Time MCARM Data Sets, http://sunrise.oc.rl.af.mil.

[4] J. Ward, *Space-Time Adaptive Processing for Airborne Radar*, Technical Report 1015, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA, 1994.

[5] *Wild-One Hardware Reference Manual 11927-0000 Revision 0.1*, Annapolis Micro Systems Inc., MD, 1997.

[6] Nikhil D. Gupta, *Reconfigurable Computing for Space-Time Adaptive Processing*, MS Thesis Proposal, TTU, http://hpcl.cs.ttu.edu/darpa/reconfigurable/, 1997.

[7] T.T. Do, H.Kropp, P. Pirsch, "Implementation of Pipelined Multipliers on Xilinx FPGAs," *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, Springer Verlag, September1997

[8] M. Morris Mano, *Digital Logic and Computer Design*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1992

[9] W. W. Smith, J. M. Smith, *Handbook of Real-Time Fast Fourier Transforms*, IEEE Press, New York, NY, 1995