Real-Time QRD-Based Beamforming on an FPGA Platform

Chris Dick
2100 logic Dr., San Jose, CA, 95124
chris.dick@xilinx.com

fred harris
College of Engineering
San Diego State University
5500 Campanile Dr. San Diego, CA 92182
fred.harris@sdsu.edu

Miroslav Pajic, Dragan Vuletic
Signum Concepts
{Miroslav.pajic, dragan.vuletic}@signumconcepts.com

***Abstract*— This paper describes the architecture, design flow and verification process for the FPGA implementation of a real-time beamformer. One of the challenges in realizing this class of processing is in the implementation of the linear algebra operations required in forming the least-squares solution to the Normal equations. We describe the FPGA realization of a flexible QRD-based approach to this problem in which the system parameters (row and column dimensions) can be supplied to the beamformer module at run-time. The design and FPGA implementation of the beamformer architecture and verification framework is described along with implementation considerations for Xilinx Virtex^{TM}-4 family of FPGAs. A model-based FPGA design flow called System Generator^{TM} [4], based on the The Mathworks Simulink®visual programming environment, was used exclusively to generate the implementation. The use of this tool chain for hardware verification is discussed. The FPGA resource utilization and performance of the QRD processor is reported.**

I. Introduction

All real-world communication systems are composed of a rich mixture processing requirements. For example, there is typically an embedded processing component, that might, for example perform functions such as link and network layer processing, including support for running a TCP/IP stack. This class of processing is naturally serviced by an *instruction set architecture (ISA)* processing resource. The physical layer of the system will contain signal processing functions in the base-band and digital IF (intermediate frequency) subsystems that are arithmetically complex and have hard real-time deadlines. For high-performance systems, an ISA processing resource is frequently not a good match for this part of the system. With its vast array of spatial computing resources, a *field programmable gate array (FPGA)* is well matched to the baseband and digital IF processing requirements of a state-of-the-art wireless communication system. With Power PPC 405 technology embedded as hard IP blocks in FPGAs like the Xilinx Virtex-4 [1] class of FPGAs, these platform-class FPGAs are heterogeneous processing platforms capable of addressing the diverse set of processing requirements in a real system. However, for various reasons (e.g. support for legacy code), it is also common to employ a processor external to the FPGA for running application code. This paper is a case study of a beamforming application that has both a software component running on a host processor and a DSP intensive task executing on the FPGA. The goals of this project were two-fold: 1) to develop a compact FPGA implementation of a QRD circuit to be used for beamforming applications and 2) utilize the *shared memory* abstraction in the Xilinx System Generator^{TM} [4] design flow to enable the interaction of the FPGA QRD module with a host application running on a processor external to the FPGA.

The paper is organized as follows. Section II provides an outline of the minimum variance distortionless response (MVDR) beamformer that was implemented using a combination of host PC and FPGA platform. In Section III a brief review of the QR decomposition (QRD) approach to least squares beamforming is described. Section IV describes the architecture and FPGA implementation of the QRD processor. Details of the *System Generator* tool chain that was used exclusively for the implementation of the FPGA part of the system are presented in Section V. Finally, conclusions are presented in Section VI.

II. MVDR Beamformer

Adaptive beamforming [2] is the application of adaptive filters to spatial signal processing. Time series collected from uniformly spaced array elements are weighted and summed to form the signal component from a selected direction of arrival while suppressing signal components from other directions of arrival. When the directions of arrival of the undesired signal

1-4244-0785-0/06/$20.00

components are unknown or vary with time, the filter weights must be adaptively adjusted to steer nulls to their directions. The structure of the spatial processor for a linear array of spatial elements is shown in Figure 1.

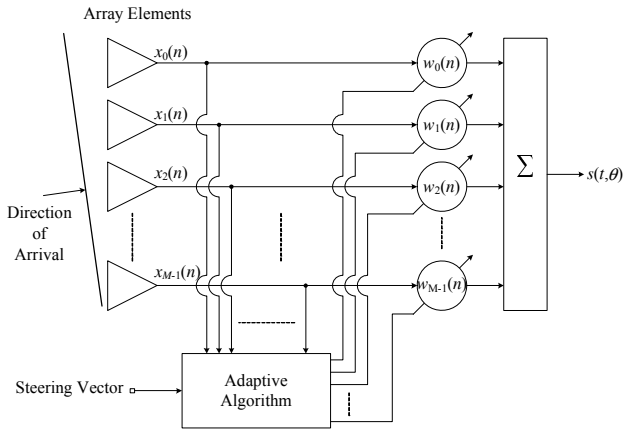

Fig. 1. Adaptive beamformer structure.

The adaptive process forms a set of filter coefficients that directs the spatial main lobe to the signal of interest while directing nulls to the undesired interfering signals. The adaptation process is performed subject to a constraint that the steering vector has unity gain in the signal direction. The steady state weights of such a beamformer form the minimum variance distortionless response (MVDR) from the array elements [2]. The beam forming process of extracting the time signal component from a selected direction while suppressing signals arriving from other directions is the same as the process performed by a tapped delay line FIR filter extracting a time series from a selected frequency band while cancelling signal components in other frequency bands contained in a single time series. The adaptive beamforming problem is called a sidelobe canceler while the tapped delay line filter is called a line canceler.

For reasons of numerical robustness and computational complexity, a common method for computing the required weight vector without directly inverting the correlation matrix is based on QR decomposition [2], and this is the approach adopted here. Readers are directed to [2] for details of the procedure.

III. The QRD Matrix Inversion Process

The QRD process is formed by a sequence of two operators [2]. These are the unitary rotations that convert complex input data to real data and an associated angle and element combiners that annihilate the selected elements of the input data set one by one. The QRD process is most compactly represented in the signal flow diagram of Figure 2. This representation is the systolic array realization of the QRD least squares solution processor. The array contains three types of processing cells. These are the boundary cells, internal cells, and output cell. The boundary cells perform the "vectoring" operation on complex input samples to nullify their imaginary parts and form rotation angles used by the internal cells. The internal cells perform Givens rotations [2] of the input values by the angles passed from the boundary cells to annihilate the non upper triangular entries of the transformed data matrix. The output cells in the linear array process the elements of the upper triangular array to perform the required back substitution [2] to produce the beamformer weights.

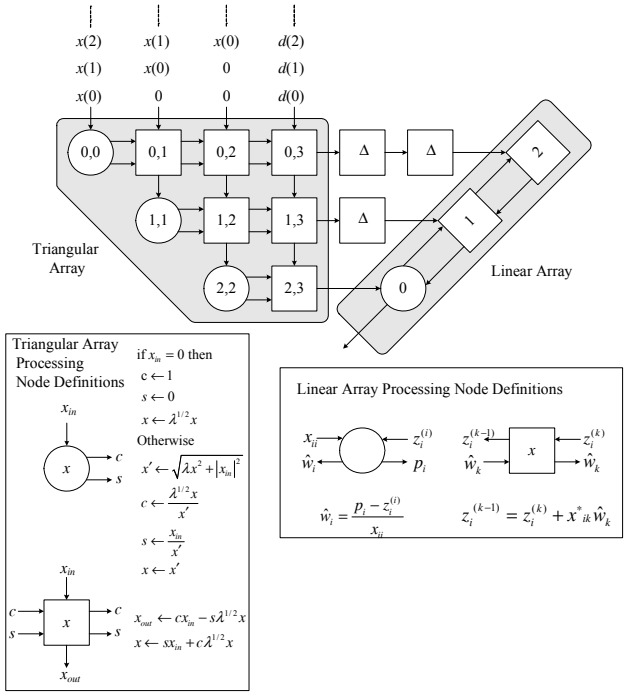

Fig. 2. Systolic array implementation of QRD matrix inversion for a 3×3 array.

IV. FPGA Implementation of QRD

The design emphasis on our implementation was on producing a compact QRD FPGA implementation. The design consists of a single boundary, internal and back substitution cells. The systolic array in Figure 2 is folded on to this set of processing resources. The boundary cell is required to compute two angles. The first angle, $\phi = \arctan(\Im(x_{in})/\Re(x_{in}))$, is used to transform the complex input samples presented to the boundary cell input port in to real-valued data. The transformation that forces the imaginary component of x_{in} to 0 must be applied to all elements in the same row associated with the boundary cell, and this operation is one of the tasks performed by the internal cells. Now that the data in the leading position of two adjacent rows are real-valued, a second angle, θ, is formed as $\theta = \arctan(x_{in} e^{-j\phi}/x)$, and is used to annihilate a term of the input data set, in an ordered manner that eventually produces the upper-right triangular matrix R. The arithmetic employed in the boundary cells could be realized in hardware by literally implementing the equations indicated in Figure 2. This would require hardware support for performing

square-roots and divisions. While these circuits are commonly implemented in FPGA hardware, we are motivated to seek alternative methods for computing the required angles that have a lower resource cost over that of the direct and obvious implementation. One well known low-complexity method for computing angles is the *vectoring* mode of the *COordinate Rotation Digital Computer (CORDIC)* algorithm [3]. The CORDIC algorithm is an iterative procedure that is capable of computing a rich set of mathematical functions. The elemental operations required in the CORDIC algorithm are addition, subtraction, bit-shift and table lookup. All of these functions are efficiently supported by FPGA architectures such as the Virtex-series of devices from Xilinx, and so the vectoring mode of the algorithm is a good candidate for the foundation of the QRD processor boundary cells. As shown in Figure 3, two CORDIC engines will be employed in the boundary cell, one for computing ϕ and the other for computing θ.

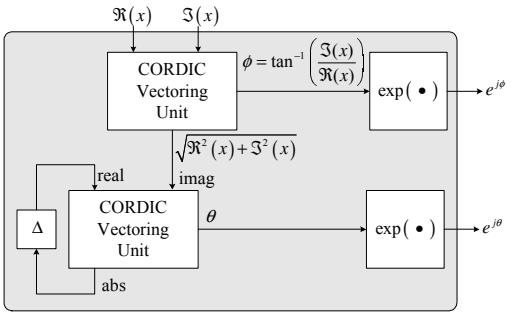

Fig. 3. Boundary cell architecture based on two vector-mode CORDIC processing engines.

Figure 4 shows the signal flowgraph for the CORDIC algorithm. After a vector of operands x_i, y_i and z_i is presented to the circuit, a series of butterfly-like cross-additions is employed to update the current estimate of the required function, with each iteration refining the estimate by approximately 1-bit of precision. For a process employing an N iteration CORDIC process, a new output will be generated every N clock cycles and a new set of operands can be presented every N clock cycles. To increase the throughput of the boundary cell, the unfolded or fully parallel architecture of Figure 5 was employed in our implementation. After the initial start-up latency of the circuit has been absorbed the initiation and completion rate of the cell is one new input/output per clock cycle.

Each data element x_{in} entering an internal cell (Figure 6) in row m must be rotated by the angle ϕ computed by the boundary cell for the m^{th} row. This task is captured in Eq. 1. We note that the variable v introduced in the equation is a temporary working variable. One option that has been used for the rotation task in QRD processors is the *rotation* mode of the CORDIC algorithm. An alternative is to simply implement the rotation in the obvious manner using multiply-accumulate (MAC) functional units, and this is the approach adopted in our implementation. The target FPGA technology for the design

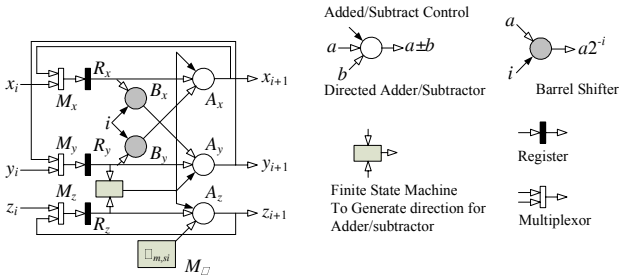

Fig. 4. Signal flowgraph for the CORDIC algorithm

is the Virtex-4 class of FPGAs [1]. These devices have a vast array of embedded MAC units referred to as the *DSP48* [1].

$$\begin{bmatrix} \Re(v) \\ \Im(v) \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \cos(\phi) & \sin(\phi) \end{bmatrix} \begin{bmatrix} \Re(x_{in}) \\ \Im(x_{in}) \end{bmatrix} \quad (1)$$

The DSP48 embedded tile supports a rich set of opcodes, capable of being updated on a per clock cycle basis, that define the arithmetic operation computed by the tile during a given clock cycle. The four multiplications implied in Eq. 1 are folded on to a pair of DSP48s, with each DSP48 computing one of the two output terms $\Re(v)$ and $\Im(v)$. Two clock cycles are required to compute the two output terms. Each DSP is supplied with a unique opcode for each clock period. Consider computing the term $\Re(v)$. During the first clock period the product $\cos(\phi) \times \Re(x_{in})$ is computed and stored in the DSP48 product or *p* register. During the second clock cycle $\sin(\phi) \times \Im(x_{in})$ is formed and subtracted from the value in the *p* register to generate the final output term. A similar sequence of computations is performed to produce $\Im(v)$. Using the DSP48 embedded blocks rather than a CORDIC-based approach for the internal cell reduces the latency of this phase of the computation and also minimizes the amount of FPGA logic fabric (lookup tables and registers) required for the implementation. Table I provides a breakdown of the area for the major functional units in the QRD implementation along with the total area of the design. The boundary cell occupies 2145 4-input LUTs [1], with the majority of this area being associated with the two 16-iteration 18-bit precision CORDIC vectoring units in the cell. The internal cell, while being more complex than the boundary cell occupies only 256 LUTs. CORDIC-based processing would have required three CORDIC rotation engines and occupy an area that would certainly be greater than the 2145 LUTs of the boundary cell. While we have exchanged LUTs for DSP48's in this case, the tradeoff is a reasonable one. The medium to high density devices in the Virtex-4 product portfolio are equipped with several hundred DSP48 embedded tiles, and using these resources to form the three rotations associated with the internal cell is a natural choice for resourcing this element of the processing.

The $cos(\phi), sin(\phi), cos(\theta)$ and $sin(\theta)$ terms required by the internal cells are computed using a simple lookup table that maps the angles ϕ and θ computed by the vectoring units in the

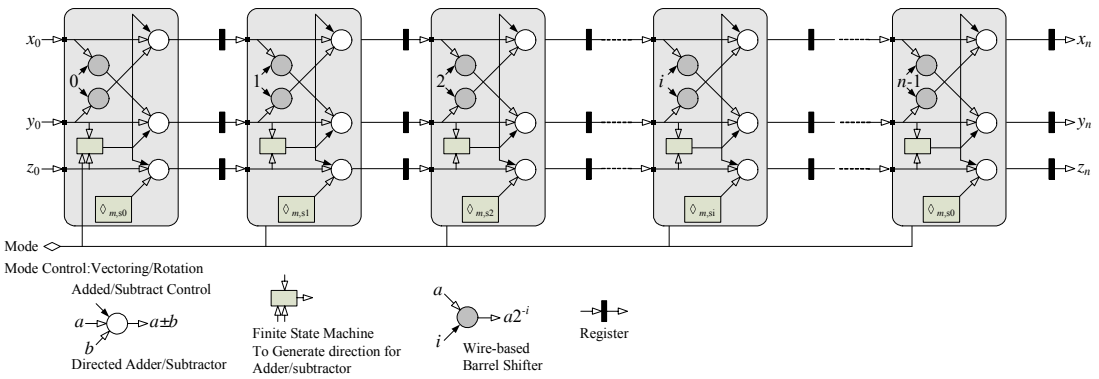

Fig. 5. Unfolded CORDIC datapath employed in the boundary cell of the QRD engine.

Functional Unit	LUTs	FFs	DSP48	BRAM	Slices
Boundary Cell	2145	2057	3	1	1266
Inner Cell	216	329	6	0	176
Back Substitution	2862	3286	4	1	1932
QRD Total	5411	5916	13	6	3530

TABLE I
FPGA RESOURCE UTILIZATION FOR FOLDED QRD AND BACK SUBSTITUTION ARRAY.

M	N	Cycles for Triangularization	Cycles for Back substitution	Total Cycles	Time (μs) for 250 MHz Clock
3	3	792	147	939	3.76
8	3	2112	147	2259	9.04
5	5	2540	255	2795	11.18
9	5	4572	255	4827	19.31
7	7	5656	371	6027	24.11
10	7	8080	371	8451	33.80
9	9	10476	495	10971	43.88
11	9	12804	495	13299	53.20
10	10	1360	560	14190	56.76

TABLE II
EXECUTION TIME FOR THE TRIANGULARIZATION AND BACK SUBSTITUTION PHASES OF THE FPGA QRD IMPLEMENTATION FOR AN $M \times N$ MATRIX.

boundary cell to their corresponding sine and cosine. Linear interpolation is applied to the output samples of the lookup table to increase the accuracy of the mapping from angle to amplitude while keeping the lookup table itself constrained to a single block RAM (BRAM) [1].

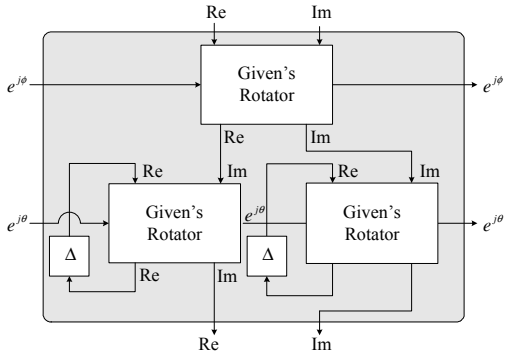

Fig. 6. Systolic array internal cell architecture employing three MAC-based Given's rotation engines.

The row and column dimensions of the input array for the QRD processor can be dynamically adjusted at run-time by simply writing the new dimensions to control registers that are part of the FPGA control plane.

Table II provides timing information for several configurations of the input data set.

V. DESIGN FLOW

The QRD implementation was realized using the Xilinx System Generator [4] for DSP model-based design flow. This is a tool chain that extends the Mathworks Simulink framework with FPGA hardware generation capabilities. System Generator is a visual design environment that allows the system developer to work at a suitable level of abstraction from the target hardware, and use the same computation graph not only for simulation and verification, but for FPGA hardware implementation. System Generator blocks are bit- and cycle-true behavioral models of FPGA intellectual property components, or library elements. The library based approach results in design cycle compression in addition to generating area efficient high-performance circuits.

In addition to providing a natural development environment for developing FPGA signal processing implementations, System Generator has a rich set of features that support the development of heterogeneous applications comprising of not only the FPGA element, but a processor. The processor could be the FPGA embedded Power PC 405 (PPC405) hard IP (intellectual property) block [1], the *Microblaze* soft processor [5] or a processor external to the FPGA. The beamformer developed for this project was partitioned between the host PC and the FPGA platform. In our implementation the host application

running on the PC might be considered more an element of the beamformer verification process (test bench), but it should be recognized that the host application could be as arbitrary and as complex as required by the task at hand. The beamformer host application is a Matlab®script (m-code) that is simulating the sensor array for the beamforming network. The script simulates a dynamic target, and generates the samples of the far-field radiation pattern for the moving target. The samples of the electric field at each sensor are generated in Matlab and forwarded to the FPGA QRD processor where a new estimate of the beamformer weight vector is produced and returned to the Matlab environment for further processing. In this case the additional processing involves plotting the polar radiation pattern for the updated complex valued weight vector. An additional point of note is that the host application does not necessarily have to be associated with the Matlab environment, and that the application could be, for example, a program written in C.

An interesting element of the beamformer application is the management of the interface between the host application, running on the PC in this case, and the QRD process executing on the FPGA platform. System Generator provides a suite of shared memory library objects (ROM, RAM, FIFO etc) that abstracts virtually all of the details of the processor-FPGA interface, and enables the host software and FPGA hardware to be somewhat insulated from each other as shown in Figure 7. For the example at hand, each new update of the beamformer is essentially a 3-step procedure: 1) new input samples from each antenna elements, as generated by the Matlab host application, need to be forwarded to the QRD engine in situ on the FPGA, 2) the QRD process is triggered, 3) the new weight vector is returned from the FPGA to the host. The shared memory library modules, and associated API (application programmer interface), transform the transfer of data between the FPGA and the host PC into to simple assignment statements based on name-space references in Matlab (or C). For example, the new weight vector `w`, resident in the Matlab workspace, is updated with the new beamformer coefficients, `FPGAWeights`, as computed by the FPGA QRD process, using the simple assignment `w = FPGAWeights`. `FPGAWeights` is the name assigned to a shared-memory buffer in the System Generator description of the QRD engine. The management of this type of host processor/FPGA interaction by the System Generator framework makes the process of developing heterogeneous applications straightforward, rapid, less error-prone, and enables an FPGA accelerator engine, like the QRD module in this case, to be easily ported between different hardware platforms without a need to modify the FPGA source code, that is, the System Generator model itself.

The interface abstraction supports transactions between the host application and the System Generator source model, as well as the host application and the final design running on the FPGA platform. This later element significantly contributes to the validation process of the software and hardware (FPGA) dimensions of the system as both components can be brought online in rapidly using the shared memory abstraction.

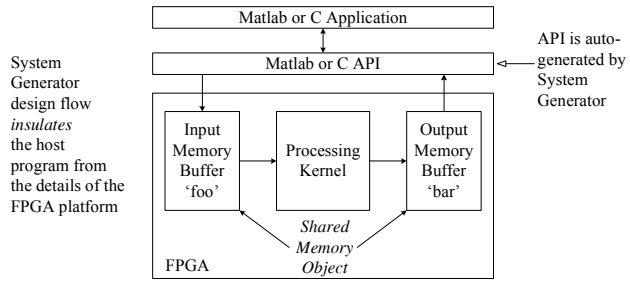

Fig. 7. Hardware/software abstraction enabled by the System Generator *shared memory* library elements. The host application performs transactions with the FPGA storage elements using simple name-space references - in this case to the memories named 'foo' and 'bar'.

VI. Conclusion

The FPGA implementation of a flexible QRD processor that enables the run-time definition of the input matrix dimensions has been described. The design employs a mixture of CORDIC-based processing (array boundary cell) and MAC-based (array internal cell) arithmetic that is well matched to the computational resources of an FPGA like the Xilinx Virtex-4 [1] family. All of the boundary cell processing and internal cell processing were projected on to a single boundary cell functional unit and internal cell functional unit, however, it should be noted that the abundant resources of FPGA platforms support the realization of a fully parallel systolic array should the throughput requirements of the target application demand extremely high-performance. In addition to the hardware architecture element of the paper, emphasis was also placed on the model-based design flow used to realize the implementation. In particular, the shared memory object capability was utilized extensively to produce a beamformer system with components running on a host PC and the DSP *heavy-lifting* supported by the FPGA. This facet of the System Generator programming environment enables the rapid development of heterogeneous systems (processors and FPGAs) while insulating the programmer from the frequently complex and error prone programming that is associated with hardware-software partitions.

VII. Acknowledgment

This work was performed by the *Xilinx Advanced Systems Technology Group (ASTG)*, the R&D organization in the *Xilinx DSP Division*, together with our partner organizations *Signum Concepts* and *San Diego State University*.

References

[1] Xilinx, "Virtex-4 user guide," *http://www.xilinx.com/xlnx/xweb/xil_publications_display.jsp?sGlobalNavPick=&sSecondaryNavPick =&category=-1210767&iLanguageID=1*.

[2] S. Haykin, *Adaptive Filter Theory, Fourth Edition*. New Jersey: Prentice Hall, 2002.

[3] J. E. Volder, "The cordic trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. 3, pp. 330–334, Sept. 1959.

[4] Xilinx, "System generator for dsp," *http://www.xilinx.com/ise/optional_prod/system_generator.htm*.

[5] Xilinx, "Microblaze soft processor core," *http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze*