Compressive Sampling Real-time Scalable Radar Signal Reconstruction Core

Michele Barbato, Gian Carlo Cardarilli, Marco Re, Ilir Shuli
Department of Electronic Engineering
Second university of Rome "Tor Vergata"
Rome, Italy
Email: shuli, cardarilli, re @ing.uniroma2.it
michele.barbato@mail.it

Filippo De Stefani, Francesco Peluso, Valerio Tocca
Engineering dept
SELEX Electronic Systems
Rome, Italy
Email: fdestefani, fpeluso, vtocca @selex-es.com

***Abstract*— The recently introduced Compressive Sampling (CS) technique can be used for reducing the speed requests for the ADC in modern radars. CS algorithm includes a reconstruction phase that is very critical in terms of real-time requirements. This paper presents a scalable hardware reconstruction core implemented in FPGA technology, aimed at real-time CS signal reconstruction.**

I. Introduction

Nowadays, for radar systems it is both dramatically important to be able to process signals with large instantaneous bandwidths, in order to achieve outstanding range resolutions, and to be able to sample at high RF frequencies, in order to avoid the costs and the SNR degradations related to mixer based conversions. In order to fulfill these requirements, fast analog to digital converters ($> 1Gsps$) and clever DSP algorithms are needed in order to manage the huge amount of samples generated. It is however difficult to design and realize fast sampling ADCs, maintaining good SNR (directly related to the number of bits of the ADC) and SFDR performances. Traditional radar scenarios present some characteristics that allow the use of the theory of compressed sensing [1] to solve the above issues. Compressive Sampling (CS) techniques could thus be introduced in order to dramatically reduce the number of samples needed to reconstruct the echo signal, reducing the performance required for the ADCs . Consequently, slower ADCs could be used to properly reconstruct the echo signal received.

An important part of the compressive sampling technique is the signal reconstruction phase. Its complexity, that results in a extremely heavy digital signal processing, makes this task very critical for an effective implementation of CS. In addition, many applications have tight real-time requirements. Among these applications, we will consider the radar systems, where each range cell must be processed in few μs.

However, only a small number of papers are present in literature regarding hardware CS signal reconstruction, while most of the papers use off-line software processing. None of the reference papers [2], [3] addresses real-time implementations. They use FPGA (or ASIC) implementations to recover CS images and evaluate the maximum clock frequency the system can run. However none of the authors indicates how to modify the core and what could be the performance in the case of larger CS systems.

There exist many signal reconstruction algorithms. The most common are: L1-minimization and Greed Orthogonal Matching Pursuit [4]. The latter of the two has lower reconstruction performance but it is more suitable for hardware implementations. Moreover the Greed OMP algorithm can have a deterministic stopping criterion which makes real-time implementations feasible.

The Greed OMP algorithm implementation can be divided into two phases: dictionary atom identification and coefficient value calculations. The first phase needs a great number of multiplications while the second one presents serial operations that can be easily executed by a MPU. The authors have created a scalable signal reconstruction core implementing this algorithm. The core can be tailored to the specific sizes of the system, taking into account the dictionary size and the sparsity level. Moreover, in this work also a fixed-point dynamic analysis was performed and the sensitivity to the data truncation has been evaluated for each algorithm phase. Fixed-point analysis is very important in FPGA implementation, since high-performance architectures cannot be easily implemented using floating-point units. In general floating-point units are very resource hungry and often they don't are the unique solution for preserving the algorithm accuracy.

The reconstruction core has been implemented in a FPGA and the performance has been evaluated. The matrix inversion is based on the QR decomposition method. This method brings great benefits in terms of speed without affecting the system performance, see [3]. Finally the paper evaluates the obtainable radar system performance and analyzes how to scale the system using bigger FPGA chips.

Partially supported by Finmeccanica SpA.

II. Greed OMP algorithm

In compressive sampling, sparse signals are represented as:

$$f = \sum_{i=1}^{n} x_i \psi_i \tag{1}$$

where $\mathbf{f}$ indicates the signal of interest, ψ indicates the representation basis and $\mathbf{x}$ is the coefficient sequence. Measured values are taken according to:

$$y_k = \langle f, \phi_k \rangle, k \in M < N \tag{2}$$

To find the non-zero values of vector $\mathbf{x}$ the equation

$$y = R\Phi\Psi x \tag{3}$$

must be solved, where R is a $m \times n$ matrix that randomly extracts M rows from Ψ. Equation 3 can be written as $y = Ax$ where $A = R\Phi\Psi$. The Greed OMP algorithm is described below.

1) *Initialize the residual R = y, the index set $\tilde{A} = \emptyset$ and the iteration counter t = 1*
2) *Find the index λ_t which is most correlated to A by solving the optimization problem*

$$\lambda_t = arg \max_{j=1..N} |\langle R_{t-1}, A_j \rangle| \tag{4}$$

3) *Update the index set Λ_t and column set $\tilde{A}$*

$$\Lambda_t = \Lambda_{t-1} \cup \lambda_t \tag{5}$$

$$\tilde{A} = \begin{bmatrix} \tilde{A} & A_{\lambda t} \end{bmatrix} \tag{6}$$

4) *Calculate the new residual according to*

$$R_t = R_t - (\tilde{A}_t \cdot \tilde{A}_t^{\,T}) R_{t-1} \tag{7}$$

5) *Increment t and return to step 2 if it is less than m*
6) *Solve the least squares problem to find $\hat{x}$ for the indices in Λ*

$$\hat{x} = arg \min_{x} \|\tilde{A}x - y\| \tag{8}$$

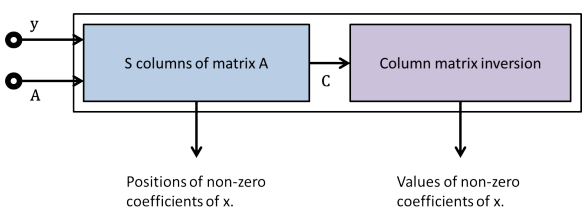

Fig. 1. Greed Orthogonal Matching Pursuit algorithm schematic.

Two different phases can be distinguished in the Greed OMP algorithm. During the first phase are identified the components (columns) of matrix A that best explain the measurement vector y. In the second phase, the columns of matrix A, identified during the first phase, are grouped together in the matrix C. The matrix C is then inverted to solve the least squares problem and calculate the values of non-zero coefficients of vector x. The two phases and their inputs and outputs are depicted in figure 1.

Taking a closer look at step 2 of the Greed OMP algorithm, it emerges that the residual vector needs to be multiplied with all the columns of matrix A at every algorithm iteration. This leads to a very big number of multiplications needed even for small dimension problems. Nowadays microprocessor units have advanced floating-point hardware multipliers but their number is generally limited (one or more in case of multicore processors, but anyway the number is very small). Even the fastest microprocessor cannot handle big dimension problems and is not able to solve the problem in real-time. GPUs, on the other side, provide plenty of hardware multipliers but they have memory bottleneck problems with these kind of systems [5]. However modern FPGA chips feature a great number of hardware multipliers. The most interesting feature of FPGAs is the fact that these chips are fully configurable and can be reconfigured for matching the specific application or algorithm. Moreover modern FPGAs have also built-in microprocessor cores, like the dual ARM cortex A9 present in the Xilinx Zynq FPGA family [6]. It is a full featured microprocessor core with L1 and L2 level cache memories and also with a full bunch of peripherals like DDR memory controllers, SPI, USB, etc.... These kinds of FPGAs are suitable for our application since the programmable fabric (a.k.a. FPGA resources) can have direct access to the ARM core L1 cache, overcoming memory bottleneck problems. The microprocessor also includes the NEON floating-point unit, suitable for serial floating-point operations. In the next section the hardware implementation of the Greed OMP algorithm will be discussed.

III. Algorithm implementation

In this section we show the implementation of the Greed OMP algorithm discussed in the previous section. We will show how the developed core can be configured to adapt it to the specific FPGA chip and to the compressive sampling problem at hand. The authors decided to implement only the first phase of the Greed OMP algorithm in hardware, leaving the second phase to be executed by the ARM microprocessor core. This choice depends on the processing characteristics: during the first phase the algorithm executes operations that can benefit from parallel hardware, like the correlation between the A matrix and the Residual Vector. During the second phase, a least squares problem must be solved. The solution is based on matrix inversion operation which is a serial task, for which MPUs are highly optimized.

The implemented hardware architecture for the first phase is shown in figure 2. The inputs are the measurement vector Y and the matrix A. Both these data are stored in memories with serial inputs. This solution allows to directly attach the memories at the peripheral bus of the MPU. These two memory blocks are custom designed for this purpose as they have serial input for memory loading but parallel output in order to speed-up the elements multiplication between the A matrix columns and the measurements vector Y, during

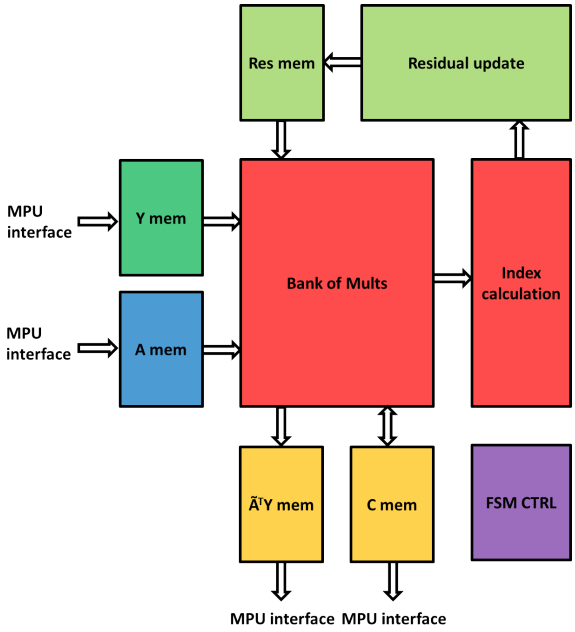

Fig. 2. Greed OMP first phase implementation.

the first iteration, and the residual vector, subsequently. The residual memory is separated from the measurement vector memory Y since the core outputs also the $\tilde{A}^T Y$ product. This value is an important factor and it required for solving the least squares problem during the second phase. It is calculated more efficiently in hardware. Moreover the core also outputs the $C = \tilde{A}^T \tilde{A}$ matrix.

The most important hardware element for the implementation of the first phase of the algorithm is represented by the *Bank of Mults* block in figure 2. It is composed of configurable hardware multipliers that perform several multiplications in parallel at each clock cycle. The *index calculation block* finds the column of A that has the maximum correlation with the residual. The output of this block is used to update the residual for the next algorithm iteration. Finally, there is a finite state machine block that triggers the various operation timing and controls the overall core functionalities.

The proposed core is configurable although the configuration cannot happen at system run-time. The hardware implementation is described by a set of VHDL files. These files can be configured using scripts. Changing some parameters in these scripts changes the system parameters such as A matrix dimensions, which are directly related to the compressive sampling problem N and M parameters. Another parameter that can be changed is the number of bits used for the representation of vector Y and the entries of the matrix A. This representation is always fixed-point, since the input values come from some hardware devices, such as an fixed-point analog to digital converter.

Using these scripts the system can be configured by changing just few parameters. The limitation of our system resides in the fixed structure of the multiplier bank, the number of used multipliers has to be always equal to the number of elements of the measurement vector Y and the bank can use only multiplier embedded in the FPGA chip. This second constraint is not a very strict limitation since there are FPGAs on the market with a huge number of multipliers, such as some Virtex 7 FPGA chips that feature 3600 hardware multipliers.

Phase two of the Greed OMP algorithm is implemented into the ARM core. In this case data are represented in floating-point format. The scope of this representation is twofold, in this case the core doesn't take speed advantages by using fixed point multiplications and, moreover, the procedure of matrix inversion tends to be unstable if a very large fixed-point wordlenght is not used. The implementation on the

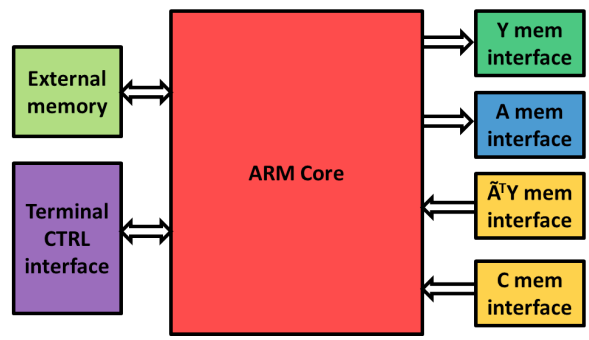

Fig. 3. Greed OMP second phase implementation.

ARM core is shown in figure 3. Basically the ARM core has two interfaces toward the Y vector memory and the A matrix memory for loading. It has also memory interfaces for the $\tilde{A}^T Y$ and C memories. The linux operating system, that runs on the processor, sees these memories like separated peripherals and not as a hardware accelerator. The usage of the operating system allows for easy control of operations using a terminal interface and also allows access to external memory for input and output data storage. The program runs as depicted by the flow chart in figure 4. The processor loads the input data from external SD card into the memories storing Y and A and starts the execution of phase 1. When phase 1 is completed the hardware accelerator triggers an interrupt and the processor reads the results of phase 1 from the memories C and $\tilde{A}^T Y$. After that, the processor solves the least squares problem using QR decomposition algorithm for matrix inversion. At the end the results are stored again into the external memory for further consultation.

IV. Results

For evaluating the designed core we implemented a compressive sampling problem using $Y = 8$ measurements, expressed as complex data, and an 8 by 64 dictionary matrix A with complex values as well. The implementation results that are shown in table I. The sparsity of the problem is user definable, from 1 to 4 in this particular case, but the core doesn't have limits regarding this parameter. The core for phase 1 has an input port for the sparsity definition, before the elaboration start. In phase 1 core runs at a maximum frequency

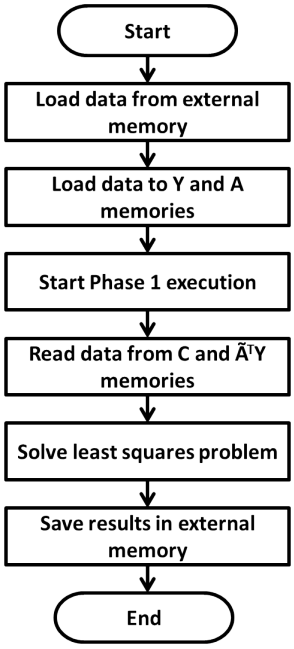

Fig. 4. Implemented system flow chart.

TABLE I
HARDWARE ACCELERATOR RESULTS.

Compressive Sampling Problem	
Variables	Complex
M	8
N	64
Sparsity	Run-time definable
Phase 1 hardware implementation	
Data	Complex
F_{MAx}	216 MHz
Mults	34
LUTs	28,000

of 216 MHz into the ZC7Z020 device and it uses 34 embedded multipliers and about 28,000 LUTs. The number of multipliers (34) is related to the complex definition of the data. This corresponds to 4 multipliers for every complex multiplication. Though it is possible to use only 3 multipliers for a complex multiplication the authors chose the 4 multipliers implementation for performance purposes. This brought the core to run at more than 200 MHz whereas the fastest FPGA implementation in literature is 85 MHz [3]. The number of LUTs occupied by the phase 1 accelerator is high but this number doesn't scale linearly with the dimensions of the compressive sampling problem. The ARM processor runs at 866 MHz, so inverting a small matrix (e.g.: 6 by 8) is not a problem. Our architecture allows pipelining so phase 1 and phase 2 can be executed in parallel.

The drawback of our present system is the use of GPIO for the implementation of the memory interfaces between the processor and the accelerator that limits the bandwidth to about 7 MB/s. However this limitation will be removed in the future implementation using more efficient interfacing protocols .

V. FINAL CONSIDERATIONS AND FUTURE WORK

In this paper the authors showed a scalable architecture for the reconstruction of compressive sampling signals aimed at real-time processing of radar signals. The architecture is faster than the FPGA state-of-the-art implementations present in the literature. The architecture aims at a clever resource usage, exploiting the parallelism of the FPGA, where it gives the major benefits, and the fast sequential operations offered by microprocessors, where this kind of processing is required.

Future work will include the use of the Snoop Control Unit to speed-up communication between the microprocessor and the accelerator.

REFERENCES

[1] E. Candés, M. Wakin, *"An introduction to compressive sampling"*, IEEE Signal Processing Magazine 25(2), pp. 2130, 2008.

[2] A. Septimus, R. Steinberg, *"Compressive sampling hardware reconstruction"*, Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3316-3319, 2010.

[3] J.L.V.M. Stanislaus and T. Mohsenin, *"High performance compressive sensing reconstruction hardware with QRD process"*, IEEE International Symposium on Circuits and Systems (ISCAS), pp. 29-32, 2012.

[4] J. Tropp and A. Gilbert, *"Signal recovery from random measurements via orthogonal matching pursuit"*, IEEE Trans. on Information Theory, vol. 53, no. 12, pp. 46554666, 2007.

[5] M. Andrecut, *"Fast GPU implementation of sparse signal recovery from random projections"*, 2008. [Online]. Available: http://www.arxiv.org/PS cache/arxiv/pdf/0809/0809.1833v1.pdf

[6] Xilinx Zynq FPGA family page http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html