Computational Functions' VLSI Implementation for Compressed Sensing

Shrirang Korde, Amol Khandare, Raghavendra Deshmukh, and Rajendra Patrikar

Electronics Engineering Department, VNIT, SA Road, Nagpur-440010
{shrirang.korde,amolkhandare2}@gmail.com,
mona1810@yahoo.com, rajendra@computer.org

Abstract. Compressed Sensing (CS) is found to be promising method for sparse signal recovery and sampling. The paper proposes the architecture for computing various computational functions useful in realizing CS recovery consisting of Singular Value Decomposition (SVD) using Bi-diagonalization method; L_1 norm of vector, L_2 norm of vector calculations. This is one of the early VLSI implementation attempt for CS recovery. We have verified the design for speed and accuracy of results on FPGA.

Keywords: Compressed Sensing, Compressive Sensing, Architecture.

1 Introduction

Compressed Sensing (CS) has been receiving a lot of interest as a promising method for sparse signal recovery and sampling. As a general principle, a sparse solution x to an under-determined linear system of equations "Ax = y" may be obtained by minimizing the L1 norm of x. Minimizing ||x||1 is recognized as a practical avenue for obtaining sparse solutions x. If the "observation" y is contaminated with noise, then an appropriate norm of the residual (Ax – y) should be minimized. If there is noise in y, the L1-regularized least square problem (LSP) [1-4]

$$\text{minimize} \|Ax - y\|_2^2 + \lambda \|x\|_1 \quad (1)$$

$$\lambda > 0, \|x\|_1 \rightarrow L_1 \text{ norm}, \|Ax - y\|_2 \rightarrow L_2 \text{ norm}$$

Numerous schemes have been proposed for obtaining sparse solutions of underdetermined systems of linear equations; popular methods have been developed from many viewpoints: L1-minimization, convex regularization and nonconvex optimization [2-5], matching pursuit [2-5], iterative thresholding methods and subspace methods [6-7], Singular Value Decomposition (SVD) methods [8-10]. These specific proposals are often tailored to different viewpoints, ranging from formal analysis of algorithmic properties to particular application requirements.

As per our review, Patrick et al [7] work is an early paper proposing VLSI Implementation for CS recovery using message passing/iterative methods. Yeyang et al [10] proposes to use SVD as data-adaptive sparsity basis for compressed sensing Magnetic Resonance (MR) images and is able to give sparser representation for

M.S. Gaur et al. (Eds.): VDAT 2013, CCIS 382, pp. 35–43, 2013.

broader range of MR images as it is better than the conventional transforms like Discrete Cosine Transform and Discrete Wavelet transform.

We reviewed various CS recovery algorithms. Matrix and vector processing is most suited to implement various computational functions. The paper contributes in identifying and implementing various computational functions required for VLSI implementation. The functional verification of the VLSI implementation was done by using LAPACK/LINPACK/TNT (Template Numerical Toolkit) software wherever required.

The paper is organized as follows. The section 2 give Computational Functions, Section 3 give Design and Implementation and Section 4 give results. The conclusion and future scope is presented in section 5.

2 Computational Functions

The review of various CS recovery algorithms [1-10], highlight use of L_1 norm of vector, L_2 norm of vector, matrix SVD function as basis for implementing various CS algorithms. The current work focuses on following CS recovery algorithms: L_1-minimization, convex regularization [2-4], iterative thresholding methods [6-7] and Singular Value Decomposition (SVD) [8-10].

The computational functions are defined as follows: L_1 norm of vector = $\Sigma_i |x_i|$, L_2 norm of vector = Squareroot ($\Sigma_i |x_i|^2$) and Matrix SVD includes solving least square problem (eq. 1), i.e. computing the following equation:

$$x = \arg\min \|Ax - y\|^2 \tag{2}$$

To compute equation (2), SVD is computed as factorization of matrix **A** (= USV^T). U, S, V denote matrix factorization of **A**

There are two methods for computing the SVD: Bi-diagonal form with QR algorithm and Jacobi rotation method. The QR algorithm is computationally much more efficient than the Jacobi method. On the other hand, Jacobi methods exhibit much more inherent parallelism than the QR. The review indicates SVD FPGA implementation [11-15] using Jacobi rotation. Jacobi SVD [15] analyzes small and mid sizes matrices around 8X8 size. The Jacobi method works well for real symmetric matrices. However the algorithm is slower for matrices of order greater than about 10, by a significant constant factor, than the Bi-diagonal form with QR method [16].

Compressed sensing recovery normally deals with larger matrix sizes. We require SVD method dealing for dense, real non-symmetric matrices which is not the case with Jacobi method. The accuracy requirement is also important and requires floating point operations. We propose SVD calculation as Householder's reduction to Bi-diagonal form (Bi-diagonalization) followed with QR algorithm. The Householder's reduction [17, 18] reduces a matrix to bi-diagonal form by repeated transformation. Transformation annihilates the required part of whole column and whole corresponding row by using a Householder matrix of the form $P = 1 - 2w.w^T$.

3 Design and Implementation

3.1 Design

We designed various architectural entities and have presented the architecture. These entities are used multiple times in the CS recovery schemes. The details are as follows:

- L_1 norm of vector
- L_2 norm of vector
- SVD calculation using Bi-diagonalization of matrix with QR algorithm (Bi-diagonalization entity , Sum1, Sum2 and Squareroot entities)

The design and RTL implementation of L_1 norm of vector, L_2 norm of vector and Bi-diagonalization are given below. The QR algorithm's functional implementation has been used for testing SVD.

L_1 Norm and L_2 Norm:

The **Fig. 1** gives iterative design and RTL implementation of L_1 norm and L_2 norm for a vector. To our knowledge L_1 norm and L_2 norm architecture is not available in literature comes as our contribution.

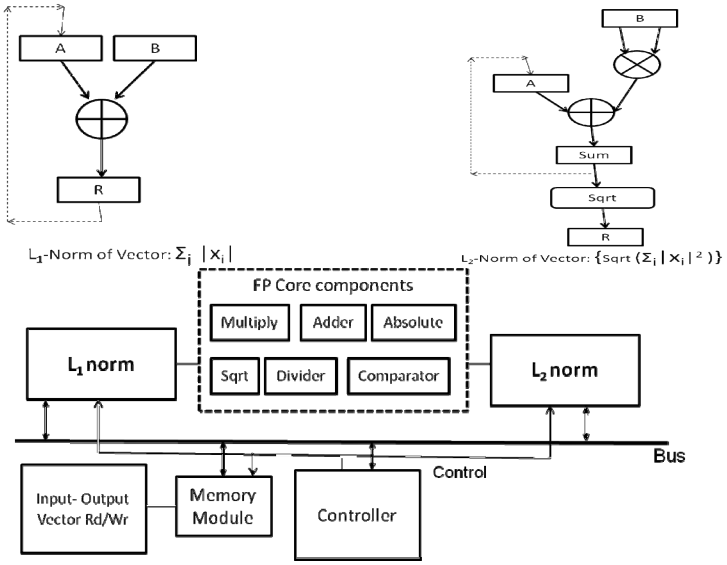

Fig. 1. (a) L_1 norm of vector (b) L_2 norm of vector (c) RTL view

Bi-diagonalization:

The Householder's reduction [17-18] to Bi-diagonal form is designed and implemented in VHDL. The Bi-diagonalization steps are given below:

1. Compute the transformation on matrix A for the ith column and place the ith diagonal in vector1, Apply transformation
2. Place the ith row of Matrix A into vector2 for the row transformation and it's calculation
3. Store the transformation in U

4. Find the ith row transformation and place the ith super-diagonal in vector2, Apply transformation
5. Store the transformation in V
6. Order the Matrix to bi-diagonal form, storing the diagonal elements in vector1 and the super-diagonal element in another vector2
7. Generate U
8. Generate V

The **Fig. 2** gives the overview of Bi-diagonalization design and **Fig. 3** gives Bi-diagonalization entity RTL view and SVD implementation. To our knowledge Bi-diagonalization architecture is not available in literature and comes as our contribution.

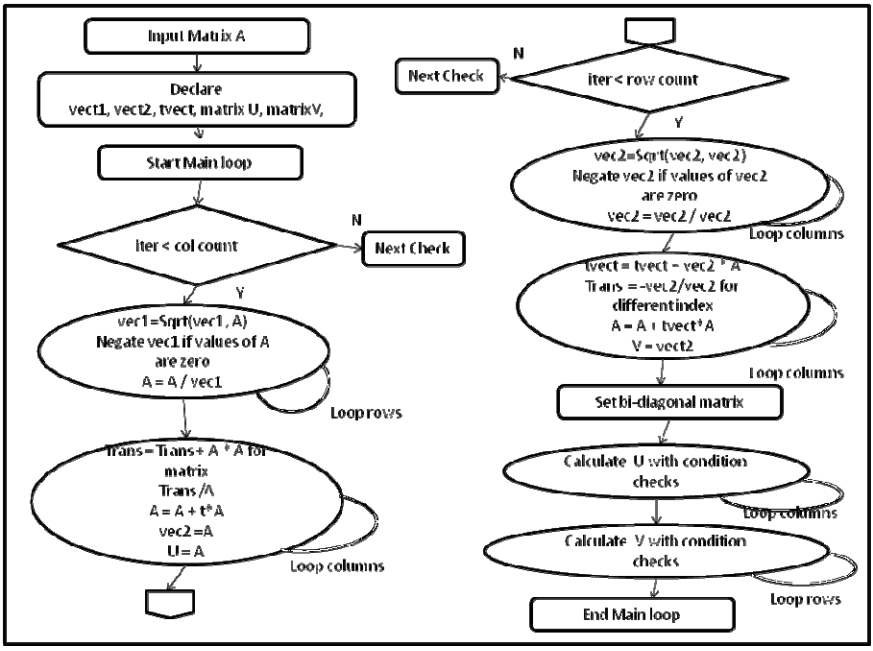

Fig. 2. Overview of Bi-diagonalization design and implementation

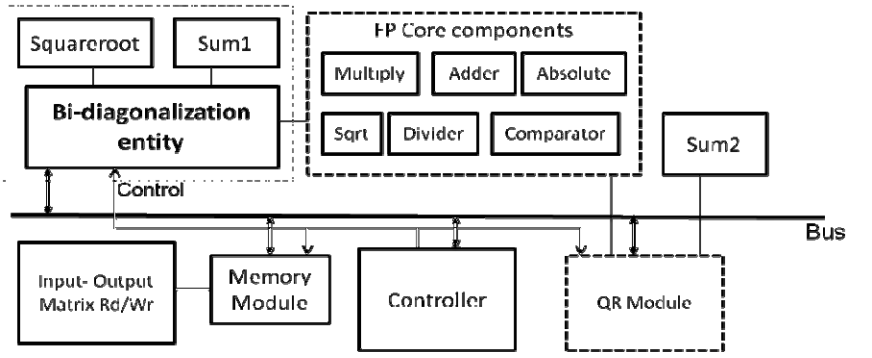

Fig. 3. Bi-diagonalization entity RTL view and SVD Implementation

The **Fig. 4** gives Sum1, Sum2 and Square root entities. The Bi-diagonalization entity (and SVD) is based on these entities. For Sum2 and Square root entities we use two multipliers (floating point) and there by introducing parallelism in operations.

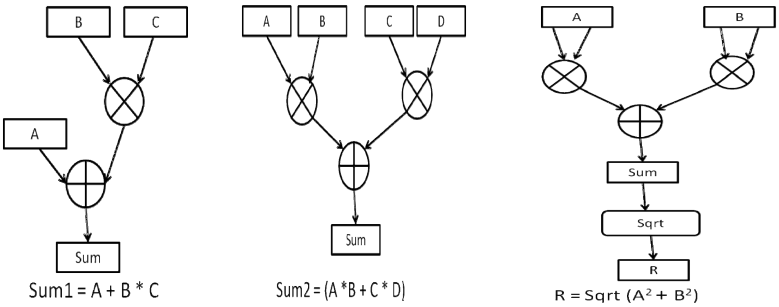

Fig. 4. (a) Sum1 (a) Sum2 (c) Square Root

3.2 Implementation

We have implemented and synthesized L_1 norm of vector, L_2 norm of vector, Bi-diagonalization of matrix, Sum1, Sum2, Square root. The Bi-diagonalization entity uses Sum1 and Squareroot entities and other FP core entities. The architectural entities are implemented and synthesized on Xilinx Artix-7 with ISE-14.x and the implementation is done in VHDL. The Floating Point core (FP) 6.0 is used.

The Xilinx FP Core (blocking mode) gives the results of calculation after certain duration. The duration is multiple (N) of clock cycles for operations like Multiply, ADD/SUB, DIV etc.

We have implemented following techniques:

- For repetitive operations, 'for loop' i.e. for functional we have provided a control flag (gated if with clock) which controls the execution to next stage. The flag is updated when all the required operations are done in each of the iteration.
- Wait for the floating point operations is implemented by using a variable to count the desired number of cycles on clock along with a flag whether to proceed to next stage or not.
- We have used above techniques for creating L1 norm of vector, L2 norm of vector, Bi-diagonalization of matrix synthesizable entities.

The **Fig. 1** gives RTL view of L_1 norm of vector and L_2 norm of vector. The memory module uses LogiCore to define memory. The controller interacts with L_1 norm entity and L_2 norm entity. The controller along with memory module gets data in/out for processing.

The **Fig. 2** gives the brief overview of Bi-diagonalization. It indicates that there are multiple loops and nested loops along with conditional checks. The design decision to create the entities like Sum1, Sum2 and Squareroot and techniques enabled us to create synthesizable entity for Bi-diagonalization.

We also have done VHDL implementation of SVD using Bi-diagonalization entity and other entities as shown in **Fig. 3**. The memory module uses LogiCore to define memory. The controller interacts with Bi-diagonalization entity and QR module. The controller along with memory module gets data in/out for processing. The QR algorithm is part of test code and uses Sum2 and other FP components. We have tested the output of SVD and verified against matlab. For the cases when rows are less than columns for a matrix A, the SVD of the A^T is to be computed initially and then perform interchanges U and V which can be part of testcode.

4 Results

The scheme has been tested using Xilinx Artix-7 with ISE-14.x and FP core 6.0. The test input given to the system was random. The device summary details are also given below for various synthesized entities.

L_1 Norm:
We have implemented L_1 norm for various vector sizes. This can be configured as per user need. We verified the output of L_1 norm using matlab as reference output and no error was observed. The time to compute L_1 norm for few vector lengths is given below and the time scales in proportion of the length.

Vector Lengths	Time
10, 100	1.305 us, 13.005 us

L_2 norm:
We have implemented L_2 norm for various vector sizes. This can be configured as per user need. We verified the output of L_2 norm using matlab as reference output and no error was observed. The time to compute L_2 norm for few vector lengths is given below and the time scales in proportion of the length.

Vector Lengths	Time
100, 49	23.695 us, 11.969 us

Bi-diagonalization:
We have implemented Bi-diagonalization and verified its output with the reference software implementation and the results were matching. The timing for a matrix size is given below and found to be acceptable for SVD implementation.

Matrix	Time
4X4	24.915 us

Device Summary for L_1 Norm of Vector and L_2 Norm of Vector and Bi-diagonalization:
Selected Device: xa7a100tcsg324-2i.

Parameters	L_1 norm	L_2 norm	Bi-diag
Number of Slice Registers*	32	97	137
Number of Slice LUTs **	32	66	76
Number used as Logic **	NA	2	12
Number of LUT FF pairs used	64	98	213
Number of bonded IOBs ***	97	97	65

(* out of 1268000, ** out of 63400, *** out of 210)
Note[1]

Design Summary of Sum1, Sum2, Squareroot:
Selected Device: xa7a100tcsg324-2i.

Parameters	Sum1	Sum2	Squareroot
Number of Slice Registers*	65	240	282
Number of Slice LUTs **	1	117	67
Number used as Logic **	1	53	67
Number of LUT FF pairs used	66	253	304
Number of bonded IOBs ***	64	161	97

(* out of 1268000, ** out of 63400, *** out of 210)

SVD:
The SVD implementation is done using various synthesized Architectural entities given above. The **Table 1** gives comparison of first diagonal element of 'S' Matrix (equation (2)) of VHDL implementation and Matlab implementation, though we have the complete Matrix (U, S, V) available, the first diagonal element being the dominant value of SVD is used for comparison in the table. We compared the timing for our SVD calculation to the implementation in [15] for size 16 X 32 and have found to be faster by a factor of 2.7. Our implementation for 32X16 takes 1.969 ms while implementation given in [15] takes 5.344 ms. The clock period used is 10 ns for getting the results but it can be reduced to 3 ns as lower limit, so the above values in the table can reduce to 1/3rd.

The Xilinx ISE value and the Matlab value indicated in the table are output of SVD for various sample input vectors. The ISE values and Matlab values are observed to be almost same and the error is negligible compared to implementation done by [15] which has error of around 1.4% for first diagonal element for 16X32 size.

[1] The device summary does not add the hardware resources used by FP core. The resource utilization for Artix7 is available in the document, LogiCORE IP Floating-Point Operator v6.2 Product Guide PG060 December 18, 2012.

Table 1. SVD results

Matrix Size	Xilinx ISE value	Matlab value	Time
4x4	4	4	41.330 us[2]
4x4	2	2	18.130 us
4X4	1.999999	2	49.750 us
32x16	16	16	1.969 ms
32x32	22.62739	22.6274	4.578 ms
96x96	67.88227	67.8823	50.904 ms
128x128	90.50966	90.5097	90.869 ms

(The timing values given above can be 1/3rd if minimum clock period of 3ns is used)

5 Conclusion

In this paper we have implemented and synthesized the computational functions required for doing compressive sensing recovery. We believe ours is one the early attempts to carry out VLSI implementation and synthesis of computational functions for compressive sensing recovery. We could not get any reference for L_1 norm of vector and L_2 norm of vector implementations and could not do comparison. Similarly, Bi-diagonalization architecture is also our contribution. Our SVD implementation has been found to be faster and more accurate compared to implementations done in [15].

We also studied algorithms (second order methods) like interior point method which can be solved in the polynomial time ($O(N^3)$) and it uses preconditioned conjugate gradient (PCG) method to approximately solve linear systems in a truncated Newton framework. Iteration methods (first order) are studied for L_1-minimization and literature has compared iterative algorithms (Hard /Soft, IST/IHT) along with its tuning. For certain very large matrices it can rapidly apply and without representing as a full matrix and in such settings, the work required scales very favorably with N. In future we plan to consider implementation of SVD based method/ first order method, along with calculation of power and present compressive sensing recovery architecture.

References

[1] Maleki, A.: Approximate Message Passing Algorithms for compressed sensing, PhD Thesis, Stanford University (September 2011)

[2] Kim, S.-J., Koh, K., Lustig, M., Boyd, S.: An Interior-Point Method for Large Scale l1-Regularized Least Squares. IEEE Journal of Selected Topics in Signal Processing 1(4), 606–617 (2007)

[3] Kim, S.-J., Koh, K., Lustig, M., Boyd, S.: An Efficient Method for Compressed Sensing. In: IEEE International Conference on Image Processing, vol. 3, pp. III-117–III-120, http://www.stanford.edu/~boyd/l1_ls/

[2] The timing corresponds to the same input that was also used for bi-diagonalization testing.

[4] Hale, E.T., Yin, W., Zhang, Y.: A Fixed Point Continuation method for l1-Regularized Minimization with Applications to Compressed Sensing, CAAM Technical Report TR07-07, Dept of Computational and Applied Mathematics, Rice University, Houstan, Texas (July 7, 2007)
[5] Baraniuk, R., Davenport, M.A., Duarte, M.F., Hegde, C.: An Introduction to Compressive Sensing. In: Connexions. Rice University, Houston (2011)
[6] Maleki, A., Donoho, D.L.: Optimally Tuned Iterative Reconstruction Algorithms for Compressed Sensing. IEEE Journal of Selected Topics in Signal Processing 4(2) (April 2010)
[7] Maechler, P., Studer, C., Bellasi, D.E., Maleki, A., Burg, A., Felber, N., Kaeslin, H., Baraniuk, R.G.: VLSI Implementation of Approximate Message Passing for Signal Restoration and Compressive Sensing. Submitted to IEEE Journal on Emerging and Selected Topics in Circuits and Systems
[8] Xu, L., Liang, Q.: Compressive Sensing Using Singular Value Decomposition. In: Pandurangan, G., Anil Kumar, V.S., Ming, G., Liu, Y., Li, Y. (eds.) WASA 2010. LNCS, vol. 6221, pp. 338–342. Springer, Heidelberg (2010)
[9] Peng, Y., He, Y.: A Reconstruction Algorithm for Compressed Sensing Noise Signal. Journal of Computational Information Systems 8(14), 6025–6031 (2012)
[10] Yu, Y., Hong, M., Liu, F., Wang, H., Crozier, S.: Compressed Sensing MRI Using Singular Value Decomposition based Sparsity Basis. In: 33rd Annual International Conference of the IEEE EMBS, Boston, Massachusetts USA, August 30-September 3 (2011)
[11] Ahmedsaid, A., et al.: Improved SVD systolic array and implementation on FPGA. In: Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology, FPT (2003)
[12] Ma, W., et al.: An FPGA based Singular Value Decomposition processor. In: IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2006 (2006)
[13] Rahmati, M., et al.: FPGA Based Singular Value Decomposition for Image Processing Applications. In: IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2008 (2008)
[14] Szecówka, P.M., et al.: CORDIC and SVD Implementation in Digital Hardware. In: IEEE 17th International Conference on Mixed Design of Integrated Circuits and Systems, MIXDES 2010, Wrocław, Poland, June 24-26 (2010)
[15] Ledesma-Carrillo, L.M.: Reconfigurable FPGA-Based Unit for Singular Value Decomposition of Large m × n Matrices. In: IEEE 2011 International Conference on Reconfigurable Computing and FPGAs (2011)
[16] Numerical Recipes- The Art of Scientific Computing, 3rd edn. Cambridge University Press
[17] Golub, Reinsch: Singular Value Decomposition and Least Squares Solutions. Handbook Series Linear Algebra, Numer. Math. 14, 403–420 (1970)
[18] Strang, G.: Linear Algebra and its Applications, 4th edn. Cengage Learning