Hardware Implementation of Floating Point Matrix Inversion Modules on FPGAs

Chetan S and Manikandan J
Crucible of Research and Innovation (CORI),
PES University, 100-Feet Ring Road, BSK Stage III,
Bangalore, Karnataka.
{email : manikandanj@pes.edu}

Lekshmi V and Sudhakar S
Control Systems Group, ISRO,
UR Rao Satellite Center (URSC),
Bangalore, Karnataka.
{email : vlekshmi@ursc.gov.in, sudhakar@ursc.gov.in}

2020 32nd International Conference on Microelectronics (ICM) | 978-1-7281-9664-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICM50269.2020.9331796

Abstract – **Matrices are employed for diversified applications such as image processing, control systems, video processing, radar signal processing, compressive sensing and many more. Finding inverse of a floating point large scale matrix is considered to be computationally intensive and their hardware implementation is still a research topic. FPGA implementation of four different floating-point matrix inversion algorithms using a novel combination of high level language programming and model based design is proposed in this paper. The proposed designs can compute inverse of a floating point matrix up to a matrix size of 25×25 and can be easily scaled to large size matrices. The performance evaluation of proposed matrix inversion modules are carried out by their hardware implementation on a Zynq 7000 FPGA based ZED board and the results are reported.**

Keywords – FPGA; Floating point; High level language; Matrix Inversion.

I. Introduction

Matrices are employed to solve diversified engineering problems with inverse of a matrix considered as an important and computation intensive matrix operation. The most commonly used algorithms to compute inverse of a large scale matrix include Gauss Jordan [1], QR decomposition [2], Singular Value Decomposition (SVD) [3] and Recurrent Neural Network (RNN) [4]. Literature survey confirms that research is in progress towards FPGA implementation of these algorithms. FPGA implementation of Gauss Jordan matrix inversion using memory optimized architecture is proposed in [5], using hardware efficient architecture with parallelism through pipelining in [6], using reconfigurable architecture in [7]. FPGA implementation of QR decomposition algorithm using hybrid segmentation technique is proposed in [8], using parallel structure for high performance in [9], using novel hardware architecture for high throughput in [10]. FPGA implementation of SVD method using customized hardware with Coordinate Rotation Digital Computer (CORDIC) is proposed in [11], using pipelined hardware architecture to accelerate SVD computation in [12]. FPGA implementation of RNN method for matrix inversion is not yet reported in the literature to the best of our knowledge and the proposed work would be the first one reporting the same.

FPGA implementation of algorithms to find inverse of a matrix, using different programming techniques are also reported in literature, which includes hardware implementation using Verilog–HDL in [8], using VHDL in [13], using model based programming in [14] and using System-on-chip (SoC) in [8].

In this paper, a novel attempt is made to program FPGAs using a combination of high level programming language and model based programming, which provides the benefit of both worlds. This enables designers with minimum or nil knowledge of hardware description languages to program FPGAs for their requirements. The proposed approach was employed to design, implement and evaluate the performance of system designed for four floating point matrix inversion algorithms (Gauss Jordan, QR decomposition, Singular value decomposition and Recurrent Neural Network method) on a Zynq 7000 FPGA.

II. Algorithms Employed for Inverse of a Matrix

A brief overview on mathematics related to finding inverse of a matrix using four different algorithms employed for proposed work is reported in this section.

A. Gauss Jordan Algorithm

Computation of matrix inverse using Gauss Jordan algorithm is illustrated in Fig. 1. Consider an $N \times N$ matrix, denoted as **A**, whose inverse needs to be computed. An identity matrix of size $N \times N$, denoted as **I**, is first appended to matrix **A**, as shown in Fig. 1, leading to a matrix size of $N \times 2N$. Elementary row operations are performed on all rows of the appended matrix using

$$R_n \leftarrow R_n - \frac{R_m}{PE_m} R_n \qquad \text{for } n=1,2,3\ldots N \quad \text{and } m \neq n \qquad (1)$$

where R_n, PE_m and R_m denote row elements of row n, pivotal element of row m and row elements of row m respectively. Pivotal elements are diagonal elements of input matrix **A**. After the completion of elementary row operations, matrix **A** will

978-1-7281-9664-0/20/$31.00 ©2020 IEEE

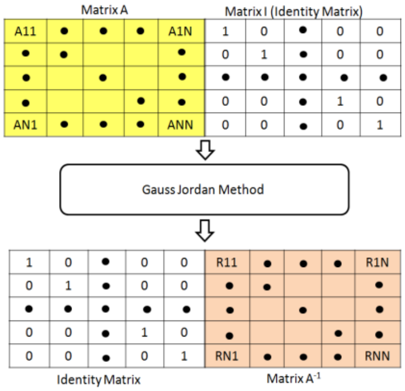

Fig. 1. Illustration of Gauss Jordan method

turn into an identity matrix and the elements of identity matrix **I** will comprise of matrix $\mathbf{A}^{-1}$ elements as shown in Fig. 1.

B. *QR Decomposition Algorithm*

QR decomposition, also known as QR factorization is a decomposition of input matrix **A** into a product of two matrices Q and R given as

$$\mathbf{A} = QR \tag{2}$$

where **A** is the input matrix whose inverse needs to be computed, R is an upper triangle matrix and Q is an orthogonal matrix. It may be noted that all three matrices are of size $N \times N$ for an input matrix of size $N \times N$. Gram-Schmidt, Householder transformation and Givens rotation are examples of techniques used to find Q and R matrix in (2). The block diagram to compute inverse of matrix **A** using QR decomposition with Givens rotation is given in Fig. 2. Matrix Q is computed as

$$Q = G_{N,1}^{T} G_{N-1,1}^{T} \cdots G_{N,N-1}^{T} \tag{3}$$

The G matrix is computed at each stage for n stages using matrices A, A_1, A_2, … A_n, where $n=N(N-1)/2$ and a sample $G_{i,j}$ matrix with size 3×3 for ease of understanding is given as

$$G_{i,j}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \quad \forall \quad i > j \tag{4}$$

where $c = \cos\theta$, $s = \sin\theta$ with

$$\theta = \cos^{-1}\left(\frac{A_{1,1}}{\sqrt{{A_{1,1}}^2 + {A_{N,1}}^2}}\right) \qquad \text{for} \qquad G_{N,1} \tag{5}$$

where $A_{i,j}$ denotes the $(i,j)^{th}$ element of matrix A, A_1, A_2, … A_n.

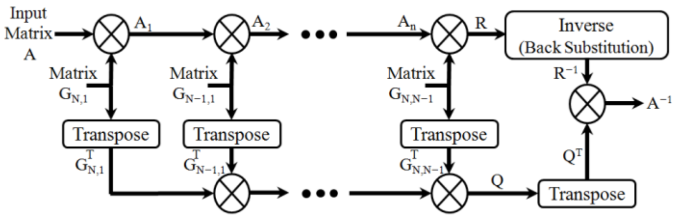

Fig. 2. Block diagram for matrix inversion using QR decomposition

Finally $\mathbf{A}^{-1}$ is computed as

$$A^{-1} = R^{-1}Q^{T} \tag{6}$$

where $R = A_n G_{N,N-1}$ and R^{-1} is computed using back substitution method.

C. *Singular Value Decomposition Algorithm*

Singular value decomposition (SVD) is a decomposition of input matrix A into a product of three matrices U, D and V given as

$$\mathbf{A} = UDV^{T} \tag{7}$$

where matrices U and V are initialized to identity matrix and are updated on iterations as

$$U_{t+1} = U_t \, G_{i,j} \tag{8}$$

$$V_{t+1} = V_t \, G_{i,j}^{T} \tag{9}$$

where $G_{i,j}$ is computed from input matrix A using (4). Even in SVD method, N(N–1)/2 number of G and A matrices are obtained as in QR decomposition. The matrix A is updated as

$$A_{t+1} = G_{i,j} \, A_t \, G_{i,j}^{T} \tag{10}$$

After a predefined number of iterations, A_{t+1} becomes more like a diagonal matrix and the same is denoted as D. Finally $\mathbf{A}^{-1}$ is computed using updated values of U, V and D as

$$A^{-1} = VD^{-1}U^{T} \tag{11}$$

with D^{-1} computed by finding reciprocal of diagonal elements in D matrix, as D converges to a diagonal matrix.

D. *Recurrent Neural Network Algorithm*

The block diagram to compute inverse of a matrix **A** using Recurrent Neural Network algorithm is shown in Fig. 3, where V_t denotes the activation state variable matrix at time *t* and η is a positive scaling constant. For proposed work, η=0.001 and $V_0=0.01\times[U]$ are considered, where U denotes all-ones matrix of size equal to the size of input matrix A. The V_{t+1} obtained at the output is expected to converge to $\mathbf{A}^{-1}$. The iterations are stopped once the error matrix E(t) given in (12) is within acceptable limits.

$$E(t) = AV_t - I \tag{12}$$

where I denotes the identity matrix. It can be observed from (12) that E(t)=0 when $V_t = \mathbf{A}^{-1}$.

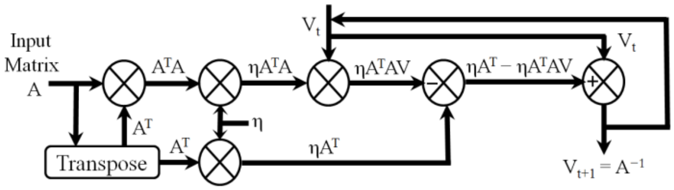

Fig. 3. Block diagram for matrix inversion using recurrent neural network

III. Hardware Implementation

All the four algorithms discussed in Section II were coded using C++ programming language. The source code in *.cpp is exported to System Generator for model based design using Vivado high level synthesis (HLS) tool. The model based design for proposed hardware implementation to find inverse of a floating point matrix with matrix size 25×25 is shown in Fig. 4. It may be noted that the same model based design is employed for all the four algorithms, by simply changing the source file for matrix inverse algorithm of interest in Vivado HLS block mentioned in Fig. 4. The hardware JTAG Cosim block in Fig. 4 is used to monitor the hardware results from FPGA in MATLAB Simulink. The snapshot of hardware setup established using ZED board to design and validate the proposed model using hardware JTAG co-simulation is shown in Fig. 5.

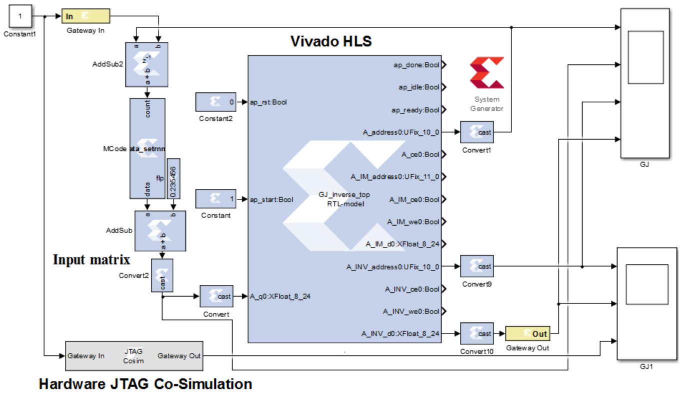

Fig. 4. Proposed model based design for FPGA implementation

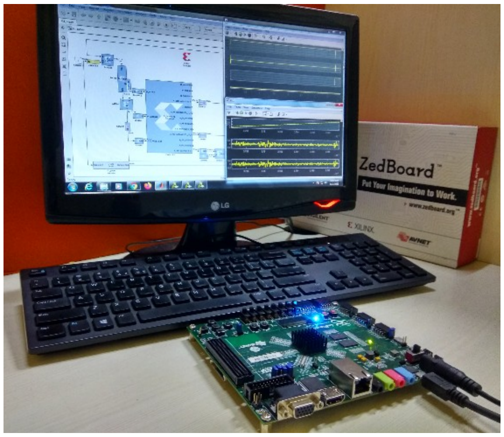

Fig. 5. Snapshot of hardware setup during JTAG Hardware Co-simulation

IV. Experimental Results

The simulation results obtained from proposed matrix inversion model using Gauss Jordan algorithm is shown in Fig. 6. In Fig. 6, input address denotes the memory address from where 625 floating point elements of the 25×25 matrix to be inverted, denoted as input data in Fig. 6, are fed as input to the model. Once the matrix inversion is completed, their values are stored in output memory given by output address. It can be observed from Fig. 6 that a valid output address is available from 3.75msec. The simulation results are compared with hardware JTAG co-simulation results and the results are shown in Fig. 7, confirming that the values of matrix inverse are identical in both simulation and hardware co-simulation. Experimental results were obtained for all the algorithms in a similar fashion.

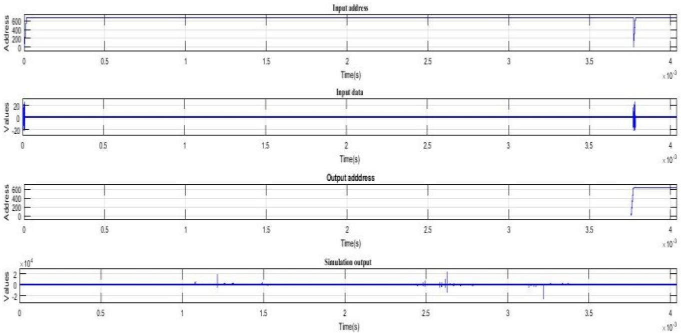

Fig. 6. Simulation results of matrix inversion module using Gauss Jordan

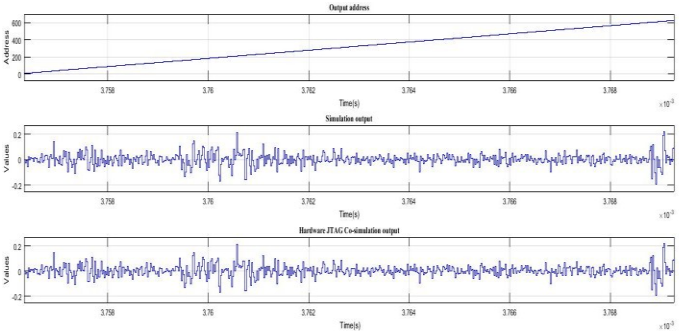

Fig. 7. Simulation results vs. Hardware Co−simulation results

A comparison between hardware resources utilized by proposed model based design with the details reported in literature for FPGA implementation of inverse of a floating point matrix with matrix size larger than 20×20 is given in Table I. As far as the proposed models are concerned, it is observed that Gauss Jordan algorithm consumed least resources followed by RNN, QR decomposition and SVD methods.

Details about power consumption, computation time to find inverse of a 25×25 floating point matrix and maximum absolute error obtained with each model are reported in Table II. It may be noted that an identical matrix was used for the fair evaluation of all the four algorithms. It is observed from Table II that QR decomposition algorithm is the fastest, followed by Gauss Jordan, SVD and RNN methods. The maximum absolute error of RNN method can be reduced by increasing the number of iterations. The error reported in Table II is for 1000 iterations on using RNN, and 10 iterations for SVD. Other two algorithms are non-recursive. It is also observed that the total on-chip power consumption for all the four models is almost equal.

TABLE I. COMPARISON BETWEEN HARDWARE RESOURCES REPORTED IN LITERATURE AND PROPOSED MODEL BASED DESIGNS

Ref	Matrix Size	Algo	FPGA	Resources Utilized
[14]	25 × 25	Gauss Jordan	Zynq 7000	LUTs: 8074, FF: 1414, LUT-RAM:1318, BRAM: 13, DSP48: 25
[14]	25 × 25	QR-D	Zynq 7000	LUTs: 8253, FF: 7430, LUT-RAM: 641, BRAM: 11, DSP48: 34
[15]	36 × 36	Gauss Jordan	Virtex5	LUTs: 8549, DSP48: 22
[16]	64 × 64	DCD method	Virtex7	Slice Reg: 13440, RAM:512, LUTs: 29696
[17]	120 × 120	Gauss Jordan	Virtex5	LUTs: 9785, DSP48: 160, LUT-FFs: 8469
Prop. work	25 × 25	Gauss Jordan	Zynq7000	LUTs: 2209, LUTRAM: 29 FF: 1801, BRAM: 3, DSP: 5, IO: 9
Prop. work	25 × 25	QR-D	Zynq7000	LUTs: 7143, LUTRAM: 636, FF: 6741, BRAM:10, DSP:32, IO:9
Prop. work	25 × 25	SVD	Zynq7000	LUTs: 11340, LUTRAM: 659, FF: 9313, BRAM:14, DSP:49, IO:9
Prop. work	25 × 25	RNN	Zynq7000	LUTs : 2140, LUTRAM : 51 FF: 1826, BRAM: 9, DSP:17, IO:9

DCD : Dichotomous Coordinate Descent Algorithm

TABLE II. PERFORMANCE EVALUATION OF PROPOSED MODEL BASED DESIGNS

Algorithm Employed	Computation Time	Power Consumed	|Max Error|
QR Decomposition	1.05ms	124mW	2.27e–07
Gauss Jordan	3.91ms	123mW	1.56e–06
SVD Method	11.24ms	126mW	3.09e–07
RNN Method	238.6ms	123mW	1.94e–01

V. CONCLUSION

In this paper, a novel attempt is made to use a combination of high level language and model based design for FPGA implementation of floating point matrix inversion modules. Four different algorithms were employed and a comparison between their performances is reported. It is observed that the computation time to find inverse of a floating point 25×25 matrix ranges between 1ms – 238ms depending on the algorithm used. The proposed designs had negligible error, decent resource utilization and power consumption. The proposed designs are capable of finding inverse of a matrix up to a size of 25×25 and can be easily scaled up to larger matrices by simply changing the source code in high level language. The proposed approach is a boon to designers with minimum or nil knowledge of hardware description language, enabling easy and rapid implementation of designs onto FPGA based systems using high level language programming.

ACKNOWLEDGEMENT

The proposed work is funded under RESPOND (Sponsored Research) program by Indian Space Research Organization (ISRO), Department of Space, India, vide project grant No. 801/18–19, titled “Study, Design and Development of Model based programming for FPGA based system design”. The authors would also like to thank the RESPOND team members Dr. MA Paul, Shri K Renji and Shri S Ganesan.

REFERENCES

[1] Peter B, E Pablo, S Enrique and R Alfredo, “Using hybrid CPU–GPU Platforms to accelerate the Computation of the Matrix Sign Function,” Parallel Processing Workshops. Euro-Par 2009. LNCS, Vol. 6043, Springer, Berlin, Heidelberg, pp. 132-139.

[2] Lei Ma, D Kevin, McAllister and McCanny, “QR decomposition based matrix inversion for high performance embedded MIMO receivers,” *IEEE Trans. Sig. Proc.*, Vol. 59, No. 4, Apr 2011, pp. 1858–1867.

[3] H Leslie, Handbook of Linear Algebra, Ch:45 Computation of the Singular Value Decomposition. Taylor and Francis Group, 2007.

[4] Jin, Li and Hu, “RNN Models for Dynamic Matrix Inversion: A Control-Theoretical Perspective,” *IEEE Trans. Ind. Inf.*, Vol. 14, No. 1, Jan. 2018, pp. 189-199.

[5] Matos and Neto, “Memory Optimized Architecture for Efficient Gauss-Jordan Matrix Inversion,” *Southern Conf. on Prog. Log.*, Mar del Plata, Argentina, 26 – 28 Feb2007, pp. 33–38.

[6] Cheema, Nash, Ansari and Khokhar, “InvArch: A hardware eficient architecture for Matrix Inversion,” *IEEE Int. Conf. Computer Design*, New York, 18 – 21 Oct 2015, pp. 180–187.

[7] Matos and Neto, “On Reconfigurable Architectures for Efficient Matrix Inversion,” *Int. Conf. Field Prog. Log. and App.*, Madrid, 28 – 30 Aug 2006, pp. 1–6.

[8] J Vázquez, A Castillo, A Carrasco, G Longoria and A Ortegón, “FPGA-Based Hardware Matrix Inversion Architecture Using Hybrid Piecewise Polynomial Approximation Systolic Cells,” *Electronics* Jan 2020, Vol. 9, No. 1, pp. 1–14.

[9] L Martin and P Bogdan, “High-Performance QR decomposition for FPGAs,” *ACM/SIGDA International Symposium on Field – Programmable Gate Arrays*, Monterey, CA, Feb 2018, pp. 183 – 188.

[10] D Lee, A Hagiescu and D Pritsker, “Large-Scale and High-Throughput QR Decomposition on an FPGA,” *Int. Sym. Field Prog. Custom Comp. Mach.*, San Diego, CA, USA, 28 Apr – 1 May, 2019, pp. 337 – 337.

[11] A Shiri and G Khosroshahi, “An FPGA Implementation of Singular Value Decomposition,” *Iranian Conference on Electrical Engineering*, Yazd, Iran, 30 Apr – 2 May, 2019, pp. 416 – 422.

[12] Y Ma and D Wang, “Accelerating SVD computation on FPGAs for DSP systems,” *IEEE International Conference on Signal Processing*, Chengdu, 6–10 Nov 2016, pp. 487 – 490.

[13] A Kumar, Subbareddy, Reddy, Raju and Elamaran, “An approach to design a matrix inversion hardware module using FPGA,” *Int. Conf. Cont., Instr., Comm. and Comp. Tech.*, Kanyakumari, 10–11 July 2014, pp. 87-90.

[14] S Chetan, V Lekshmi, Sudhakar S and J Manikandan, “Design and Implementation of a Floating point matrix Inversion Module using Model based Programming,” *IEEE 16th India Council International Conference (INDICON)*, Rajkot, India, 13–15 Dec 2019, pp. 1-4.

[15] García et al., “Suitable FPGA implementation of floating-point matrix inversion based on Gauss-Jordan elimination,” *South. Conf. on Prog. Log.*, Argentina, 13–15 Apr 2011, pp. 263-268.

[16] Quan and Liu, “Efficient complex matrix inversion for MIMO OFDM systems,” *Jour. Comm. Net.*, Vol.19, No.6, Dec. 2017, pp. 637-647.

[17] García et al., “FPGA implementation of large-scale matrix inversion using single, double and custom floating-point precision,” *South. Conf. on Programmable Logic*, Bento Goncalves, 20–23 March 2012, pp. 1-6.