Architectural Design Tool for Low Area Band Matrix LU Factorization

Semih Aslan, Erdal Oruklu, and Jafar Saniie

Electrical and Computer Engineering Department

Illinois Institute of Technology

Chicago, Illinois, 60616, USA

Abstract — **An area efficient band matrix design tool is presented for computation of LU factorization. Band matrices are used in many applications such as digital signal processing, power system analysis, finite element systems, and many more. In many of these applications, LU factorizations are needed to solve linear equations and calculation of a matrix determinant. Due to the large amount of data that needs to be computed and stored for this factorization, the goal of the proposed design tool is area minimization without compromising speed. Based on the number of zero elements and bandwidth, the design tool uses a High Level Synthesis to create the RTL code and testbench for these factorizations for the desired matrix. The proposed design tool reduces area, required memory location and chip to market time by generating a testbench and doing error analysis during RTL code generation. The generated RTL code is universal and can be used directly for any FPGA and VLSI platforms.**

I. INTRODUCTION

There are many physical systems that can be implemented as matrices containing few non-zero elements. Often non-zero elements are less than 10% of the total elements in a matrix [1] and data is said to be sparse. Sparse matrices are used in applications such as computational fluid dynamics, circuit simulations of electrical power systems, computer vision, tomography, structural system design and testing, finite-element methods and many more areas [2] [3]. These applications require a massive amount of data collection, data storage and data processing. Hence, sparse matrices play an important role in reducing the number of processing elements for large matrices. The comparison between matrix nonzero elements of an ordinary matrix and a sparse matrix is shown in Figure 1.

It is clearly shown in Figure 1 that ordinary matrix operations become slow and area inefficient for large matrix sizes. In this paper, a hardware design system is presented for sparse matrices wherein non-zero elements are formed diagonally. This is known as a band or banded matrix. The general form of a band matrix is shown in Figure 2. The band matrix column length k and row length p are important parts of determining the matrix operations.

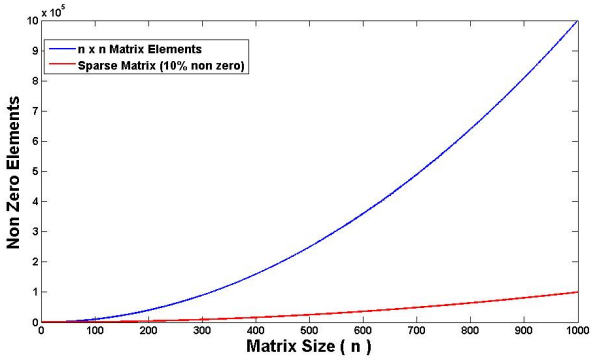

Figure 1. Number of non-zero matrix elements comparison

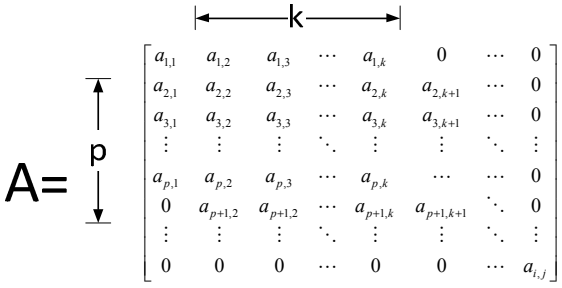

Figure 2. Band matrix

A matrix is said to be a band matrix when the conditions below are met.

$$a_{i,j} = 0 \quad if \quad j < i - p \quad or \quad j > i + k \tag{1}$$

The non-zero elements in a band matrix depend on the matrix size n, values of k and p. The calculation of the number of non-zero elements (N_{NZ}) in a band matrix can be calculated using equation (2) below.

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.

$$N_{NZ} = n + \sum_{i=1}^{k}(n-i) + \sum_{j=1}^{p}(n-j) \quad (2)$$

Another characteristic of a band matrix is its bandwidth (BW). The BW can be calculated using equation (3).

$$BW = k + p + 1 \quad (3)$$

The number of nonzero elements, matrix size n and BW are important parameters to determine the hardware size, memory size and throughput of the design. The proposed design tool uses these parameters to create the RTL code. This RTL code is Verilog HDL that is checked by LEDA [4], which optimizes this RTL code to implement on any FPGA and VLSI platform. In this paper, verification of the design is done using a Xilinx Virtex 5 [5] evaluation board.

II. Bandwidth Reduction Algorithms

Smaller BW in a sparse matrix is important to reduce the required hardware, memory size, and determination of matrix operation outcome. A sparse matrix can be converted into a band matrix to reduce its BW. There are several algorithms used to reduce the BW in a sparse matrix such as the Reverse Cuthill-McKee (RCM) [6] [7], the Rosen [8] and Gibbs-Poole-Stockmeyer algorithms [7] [9]. A simple sparse matrix graph and bandwidth reduction using RCM is shown in Figure 3 and Figure 4. The black regions in Figure 4 represent nonzero elements.

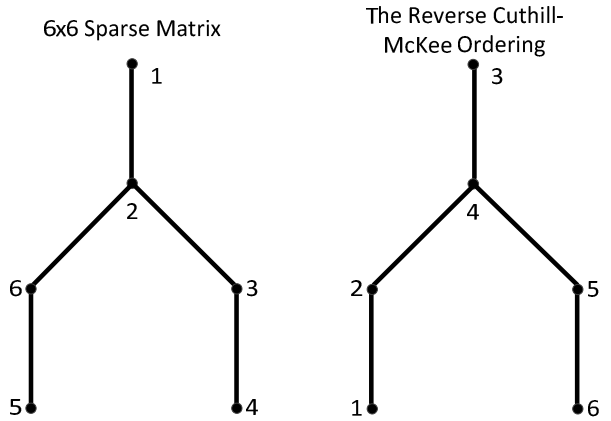

Figure 3. Graph Representation of Sparse Matrix and RCM Ordering

The main concept for these algorithms is to renumber the vertex points (or nodal points) among the diagonal elements and keep the vertex point ordering in v_{i-1}, v_i, v_{i+1} sequence. This can be seen in Figure 5 below.

Even though BW reduction and non-zero element reordering is an important topic, it is out of the scope of this paper.

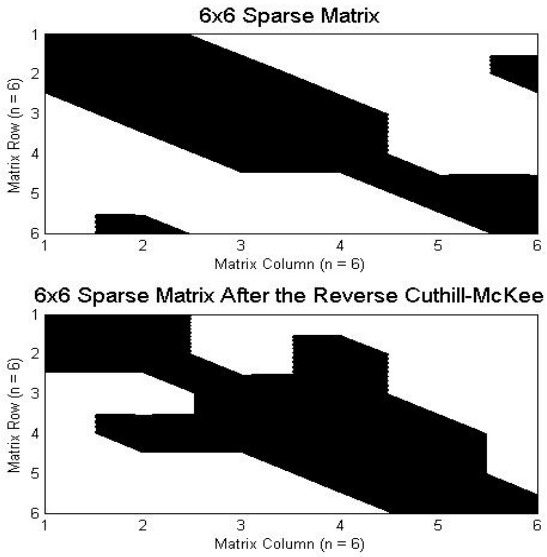

Figure 4. 6x6 Sparse Matrix and RCM Ordering

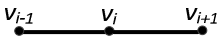

Figure 5. Bandwidth reduction reordering

III. LU Factorization

One of the most common and popular methods for solving linear equations calculating a matrix determinant is the LU factorization. An LU Factorization is where a matrix can be the represented product of lower side matrix **L** and upper side matrix **U**.

$$\mathbf{A} = \mathbf{L}.\mathbf{U} \quad (4)$$

The proposed Band Matrix LU Design System (BMDS) uses a modified LU factorization that assumes all diagonal elements are 1. The modified LU factorization with $k = p = 1$ is shown in Figure 6. This will reduce the storage elements and simplify the calculation of $u_{i,j}$ and $l_{i,j}$.

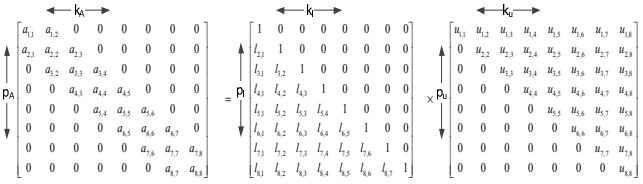

Figure 6. Modified LU Factorization

The bandwidth of **L** and **U** can be calculated based on k and p values of matrix **A**. Determining the row and column length of **L** and **U** first is important in order to reduce the hardware. This way, the BMDS only generates the hardware for the values that need to be calculated. For a given **A** matrix with row length p_A and column length k_A, row and column elements of **L** and **U** can be calculated as;

$$k_l = 0, p_l = p_A, k_u = k_A, p_u = 0 \quad (5)$$

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.

Using Figure 6 and equation (5), the values of $u_{i,j}$ and $l_{i,j}$ can be calculated.

$$u_{i,i+1} = a_{i,i+1} \tag{6}$$

$$l_{i,i-1} = \frac{a_{i,i}}{u_{i-1,i-1}} \tag{7}$$

$$u_{i,i} = a_{i,i} - l_{i,i-1} \times u_{i-1,i} \tag{8}$$

One of the usages of the LU factorization is in solving linear equations.

$$\mathbf{A}.\mathbf{x} = \mathbf{b} \tag{9}$$

Using (4) in (9), the linear equation can be written as;

$$(\mathbf{L}.\mathbf{U}).\mathbf{x} = \mathbf{b} \tag{10}$$

$$\mathbf{L}.(\mathbf{U}.\mathbf{x}) = \mathbf{b} \tag{11}$$

$$\mathbf{L}.\mathbf{y} = \mathbf{b} \tag{12}$$

By using back-substitution, the value of **y** can be calculated, and this **y** vector can later be used to calculate **x** using forward substitution.

$$\mathbf{U}.\mathbf{x} = \mathbf{y} \tag{13}$$

IV. Band Matrix LU Design System (BMDS)

The GUI of the proposed band matrix design system for LU factorization is shown Figure 7.

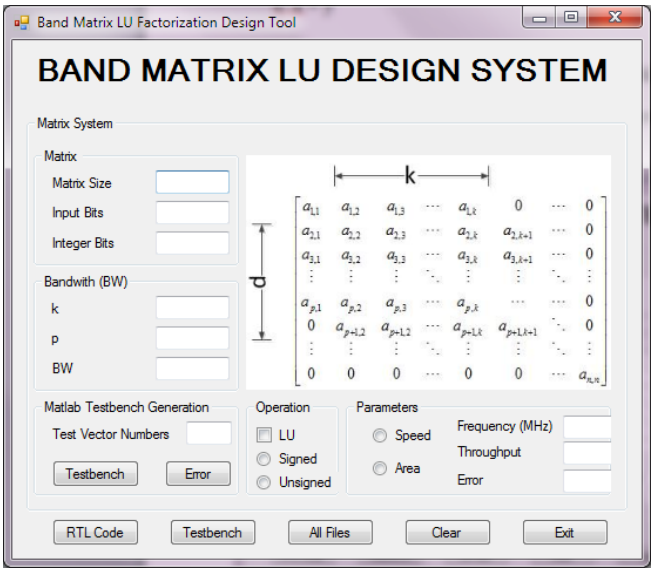

Figure 7. Band matrix LU factorization design system GUI

Based on user input parameters, the BMDS generate synthesizable RTL design and verification files. It also generates MATLAB files for error analysis and Modelsim Tcl scripting files for verification. The main program is designed using Perl and Tcl scripting languages. The design generates structural RTL code for hardware design.

The idea behind BMDS is similar to the High Level Synthesis (HLS) [10] design flow that is shown in Figure 8.

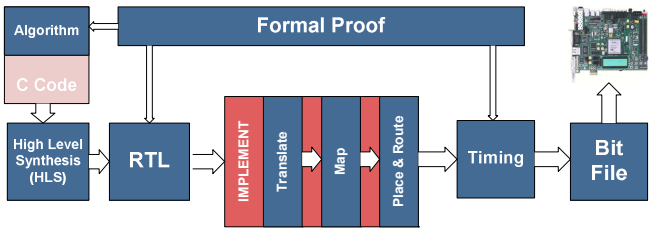

Figure 8. HLS Design flow

The BMDS is a Hybrid-HLS that uses user input parameters to design hardware rather than using algorithms. The design and verification flow of the proposed designer system is shown in Figure 9.

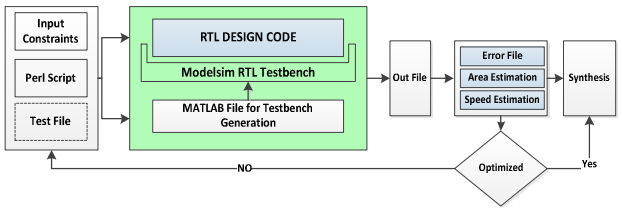

Figure 9. The BMDS design and verification flow

The BMDS system generates optimal RTL code for LU Factorization. The following input parameters are accepted by the BMDS:

- $n \times n$ square matrices where $2^1 < n < 2^{20}$
- Fixed point support for $4 < x < 128$ bits.
- Variable bandwidth matrix (k and p)
- User defined automated testbench generation
- Input constraint dependent optimization

The BMDS uses basic hardware blocks such as adders, multipliers, registers, mux, and more complicated arithmetic operation blocks such as division, square root or inverse square root operations. These more complex blocks are built into BMDS and can be used based on user selected operations. There are two algorithms which are used to design these blocks; Newton-Raphson and Goldschmidt algorithms [11].

V. Testbench Generations

One of the most important sections of the BMDS is to generate testbench files and error analysis using MATLAB and Modelsim. Before the RTL code is synthesized, it can be tested using a testbench that is created using MATLAB and Modelsim. The testbench generation and error checking block diagram is shown in Figure 10 below.

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.

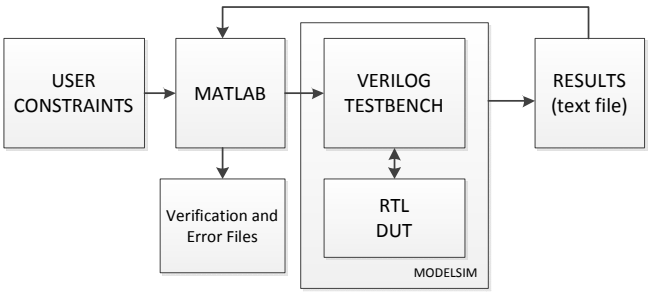

Figure 10. MATLAB and Modelsim flow

As seen on Figure 10, user constraints such as matrix size, desired operation, and input bit size from the BMDS passes to MATLAB to generate a testbench for the desired operations. MATLAB generates band matrix using the code shown below.

```
function band_matrix_1(n,k,t)
  B=rand(n);
  if ((k+t<2*n-1)& k>=0 & t>=0)
     for i=1:n
       for j=i+k:n-1
       B(i,j+1)=0;
       end
     end
     for j=1:n
       for i=j+t:n-1
       B(i+1,j)=0;
       end
     end
    B;
    else
    A='Please Enter Valid Number'
end
```

Figure 11. Band Matrix Generation in MATLAB

After generation of the band matrix using the code in Figure 11, the values in this matrix need to be converted to fixed point binary or hexadecimal values based on the design. Figure 11 generates positive random numbers [0,1]. These numbers are converted into positive and negative numbers. To generate random test values and test results, the following procedure is followed.

- Get the values for n, r, k and p
- Generate random test vectors (T)
  - For $n < 6000$ use Figure 11
  - For larger matrices generate number elements
- Generate random binary numbers using MATLAB
  - For fixed-point signed numbers:
    *T_Bin=dec2bin(T*2^n,n)*
  - For fixed-point unsigned numbers:
    *T_Bin=dec2bin(T*2^(n-i),n)*
- Generate Hex equivalent of the binary values for memory reduction using
    T_Hex=dec2hex(bin2dec(T_Bin),n/4)
- Generate the Modelsim testbench file and get results
- Compare results using MATLAB

After the generation of the band matrix in fixed point binary or hexadecimal numbers, the next step is to generate testbench and Tcl .do files that can be transferred to Modelsim. Modelsim will execute the .do file that loads the RTL code of the design (generated by the BMDS) and the testbench. After running the testbench, the results are stored in a text file. These results are later imported into MATLAB and compared by MATLAB and with the LU function that is generated for the BMDS for correct operation and error analysis. After behavioural verification of the system is complete, the RTL and testbench can be transferred into the FPGA or ASIC design flows that are explained above.

An important issue that needs to be addressed during the verification process after Modelsim generated results are imported into MATLAB is working with negative fixed-point numbers in MATLAB. This problem is addressed using the MATLAB code that is given in **Error! Reference source not found.**.

```
function [D]=bin_dec(A,k)
[m,n]=size(A);
D=zeros(1,m)';
for i=1:m
    if (A(i,1)=='1')
      m=bin2dec(A(i,:))-2^n;
      m=m/2^(n-k);
    else
      m=bin2dec(A(i,:));
      m=m/2^(n-k);
    end
 D(i)=m;
end
end
```

Figure 12. Signed and unsigned binary to decimal conversion

Matrix size becomes an important problem in design and verification due to the size of the memory in digital systems and MATLAB. For example, to store a 1024x1024 matrix, the total number of elements that need to be stored is 1,048,576. MATLAB requires 16 MB, 32MB and 64 MB to store this matrix as 8, 16 and 32 bit data, respectively. The storage that is required in MATLAB to store an n×n matrix with r-bit can be calculated using equation (14).

$$Size_{Bin-Byte} = 2\times(r\times n^2) \tag{14}$$

The graphical representation of the equation (14) shows that while testing the matrix operations for large matrices, the size of the matrix and the input data size become important issues.

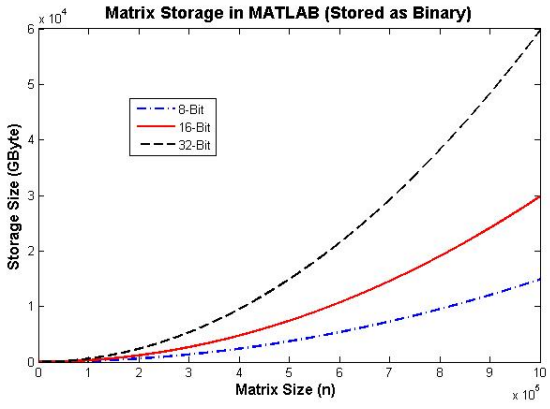

Figure 13. Matrix Storage in MATLAB (Binary Storage)

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.

To overcome this problem for large matrices, the BMDS stores only non-zero elements as hexadecimal rather than binary. The equation (14) can be written as hexadecimal storage as;

$$Size_{Hex-Byte} = \frac{(r \times n^2)}{2} \quad (15)$$

Storage of a matrix with n =1,000,000 and r = 32-Bit requires 58.2077 terabytes of storage. If this matrix is a band matrix (k=1, p=1 and only non-zero elements are stored as hexadecimal) using equations (2) and (15), the required storage can be calculated as 45.7763 megabyte. The required storage can be reduced by a factor of 1,333,333. This makes it possible to test and store for large matrices and their operations.

The BMDS accepts matrices with any size BW for n<20,000 and large matrices (20,000<n<1,000,000) with k=1, p=1, and BW=3. This is chosen due to storage problems in MATLAB. As described above, LU factorization transforms a matrix A into two matrices L and U. The size of these matrices depends on the p and k (see equation (5)) and it determines the size of memory to store matrices and testbench values.

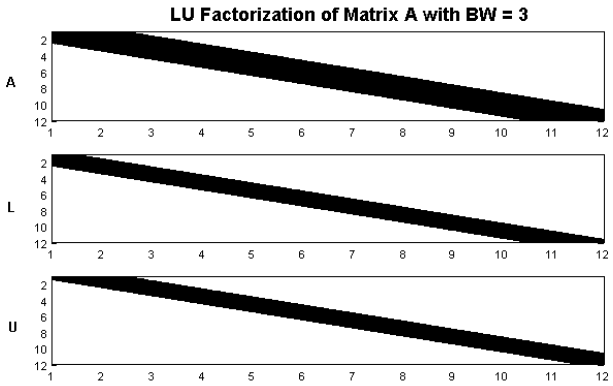

Figure 14. LU Factorization of 12x12 Matrix A with k=2, p=1

As seen in Figure 14, the LU factorization of matrix A generates L and U matrices that have BW = 2 and the same number of non-zero elements. The required memory to store matrices L and U can be calculated using equation (16) below.

$$L - Size_{Hex-Byte} = \frac{(2n-1) \times r}{2} \quad (16)$$

VI. RESULTS

The BMDS is used to generate RTL code and verification files of 16-bit, 8x8 band matrices with different bandwidth sizes. These designs are implemented using Xilinx ISE [5] tools with a Virtex-5 evaluation platform. In order to compare the BMDS results, the same LU factorization is done using Xilinx AccelDSP tool [12] for the same matrices. The area analysis of these designs is provided in Table I.

Table I. Area values of the design using Virtex 5 (r = 16-bit)

Matrix Size (n=8) (k, p)	Area			Memory Size (Kbyte)		
	BMDS	IP	(%)	BMDS	IP	(%)
(1,1)	260	1040	75	1.5	12	88
(2,1)	350	1040	66	2	12	83
(3,1)	460	1040	56	2.5	12	79
(1,2)	350	1040	66	2	12	83
(1,3)	460	1040	56	2.5	12	79

Table I shows that the smaller BW leads to lower cost hardware due to optimization used by BMDS. Another parameter that depends on the size of BW is throughput and latency. The throughput values are based on the number of matrix factorizations per second and are shown in Table II below.

Table II. Throughput values of the design using Virtex 5 (r = 16-bit)

Matrix Size (n=8) (k, p)	Throughput			Frequency (MHz)		
	BMDS	IP	BMDS/IP	BMDS	IP	BMDS/IP
(1,1)	4,100	410	10	133.3	66.1	2.02
(2,1)	3,301	410	8.1	131.5	66.1	2.02
(3,1)	2,105	410	5.1	130.0	66.1	2.01
(1,2)	3,301	410	8.1	131.5	66.1	2.02
(1,3)	2,105	410	5.1	130.0	66.1	2.01

Table II clearly shows that increasing the size of BW decreases the throughput and overall operation frequency.

VII. CONCLUSION

The values in Table I and II clearly show that using BMDS can have up to 75% area, 88% memory, and 10 times throughput savings compared IP cores. The IP core generates the design for an ordinary matrix LU factorization that has a size of an 8x8 matrix. The design that is generated using an IP core-based LU factorization becomes impractical for large size matrices due to limited FPGA and memory sizes. The BMDS and its BW oriented design structure allow designers to design an LU factorization of large matrices. In addition, memory size is drastically reduced by only storing nonzero elements in memory. This BW-oriented design system and memory storage make it possible to implement other matrix operations and factorizations on large band matrices.

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENT

The authors would like to thank Xilinx, Inc. for their valuable support.

REFERENCES

[1] A. M. Erisman, I. S. Duff, and J. K. Reid, *Direct Methods for Sparse Matrices*. New York, United States of America: Oxford University Press, 2003.

[2] A Fujii, R Suda, and A Nishida, "Parallel Matrix Distribution Library for Sparse Matrix Solvers," in *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, Beijing, 2005, pp. 219-226.

[3] Y Yun, Z Wenqing , and Y Inoue, "High-Performance Systolic Arrays for Band Matrix Multiplication," in *IEEE International Symposium on Circuits and Systems*, Taipei, May 2005, pp. 1130-1133.

[4] Synopsys. (2011, February) LEDA RTL Checker. [Online]. http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/Leda.aspx

[5] Xilinx. (2011, February) Xilinx. [Online]. http://www.xilinx.com

[6] E. Cuthill and J. McKee , "Reducing the bandwidth of sparse symmetric matrices," in *Association for Computing Machinery* , New York, 1969, pp. 157-172.

[7] S.T. Barnard, A. Pothen, and H.D. Simon, "A spectral algorithm for envelope reduction of sparse matrices," in *Supercomputing*, 1993, pp. 493-502.

[8] R. Rosen, "Matrix bandwidth minimization," in *Association for Computing Machinery* , New York, 1968, pp. 585-595.

[9] John G Lewis, "Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms," *ACM Transactions on Mathematical Software*, vol. 8, no. 2, pp. 180-189, June 1982.

[10] C. Desmouliers, S. Aslan, E. Oruklu, J. Saniie, and F.M. Vallina, "HW/SW co-design platform for image and video processing applications on Virtex-5 FPGA using PICO," in *IEEE International Conference on Electro/Information Technology*, Normal, 2010, pp. 1-6.

[11] S. Aslan, E. Oruklu, and J. Saniie, "Realization of area efficient QR factorization using unified division, square root, and inverse square root hardware," in *IEEE International Conference on Electro/Information Technology*, 2009, pp. 245-250.

[12] Xilinx. (2011, February) AccelDSP Synthesis Tool. [Online]. http://www.xilinx.com/tools/acceldsp.htm

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:04:58 UTC from IEEE Xplore. Restrictions apply.