GUSTO: An Automatic Generation and Optimization Tool for Matrix Inversion Architectures

Ali Irturk and Bridget Benson University of California, San Diego

Shahnam Mirzaei

University of California, Santa Barbara

Ryan Kastner

University of California, San Diego

Matrix inversion is a common function found in many algorithms used in wireless communication systems. As FPGAs become an increasingly attractive platform for wireless communication, it is important to understand the trade-offs in designing a matrix inversion core on an FPGA. This article describes a matrix inversion core generator tool, GUSTO, that we developed to ease the design space exploration across different matrix inversion architectures. GUSTO is the first tool of its kind to provide automatic generation of a variety of general-purpose matrix inversion architectures with different parameterization options. GUSTO also provides an optimized application-specific architecture with an average of 59% area decrease and 3X throughput increase over its general-purpose architecture. The optimized architectures generated by GUSTO provide comparable results to published matrix inversion architecture implementations, but offer the advantage of providing the designer the ability to study the trade-offs between architectures with different design parameters.

1. INTRODUCTION

Matrix inversion algorithms lie at the heart of most scientific computational tasks. Matrix inversion is frequently used to solve linear systems of equations in many fields such as wireless communication. For example, in wireless communication, MIMO-OFDM systems use matrix inversion in equalization algorithms to remove the effect of the channel on the signal [Zhou et al. 2005; Abe et al. 2003a, 2003b], minimum mean square error algorithms for precoding in spatial multiplexing [Kusume et al. 2005], and detection-estimation algorithms in space-time coding [Hangjun et al. 2003]. These systems often use a small

number of antennas (2 to 8) which results in small matrices to be decomposed and/or inverted. For example, the 802.11n standard [IEEE 802.11] specifies a maximum of 4 antennas on the transmit/receive sides and the 802.16 [IEEE 802.16] standard specifies a maximum of 16 antennas at a base station and 2 antennas at a remote station.

The computational platform plays a significant role in the overall design and implementation of wireless communication systems. A designer should determine an implementation way between a wide range of hardware: Application-Specific Integrated Circuits (ASICs) and software: Digital Signal Processors (DSPs). ASICs offer exceptional performance results at the price of long timeto-market and high NonRecurring Engineering (NRE) costs. On the other hand, DSPs ease the development of these architectures and offer a short time-tomarket, however, they lack the performance capacity for high throughput applications. Field Programmable Gate Arrays (FPGAs) strike a balance between ASICs and DSPs, as they have the programmability of software with performance capacity approaching that of a custom hardware implementation and present designers with substantially more parallelism, allowing more efficient application implementation.

FPGAs are an increasingly common platform for wireless communication [Meng et al. 2005; Iltis et al. 2006; Cagley et al. 2007]. FPGAs are a perfect platform for computationally intensive arithmetic calculations like matrix inversion as they provide powerful computational architectural features: vast amounts of programmable logic elements, embedded multipliers, shift register LUTs (SRLs), Block RAMs (BRAMs), DSP blocks, and Digital Clock Managers (DCMs). If used properly, these features enhance the performance and throughput significantly. However, the highly programmable nature of the FPGA can also be a curse. An FPGA offers vast amounts of customization which requires the designer to make a huge number of system, architectural, and logic design choices. This includes decisions on resource allocation, bit widths of the data, number of functional units, and the organization of controllers and interconnects. These choices can overwhelm the designer unless she is provided with design space exploration tools to help her prune the design space.

For more efficient design space exploration and development, we designed an easy-to-use tool, GUSTO (**G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization),

which allows us to select various parameters such as different matrix dimensions, integer and fractional bits of the data, resource allocation, modes for general-purpose or application-specific architectures, etc. [Irturk et al. 2008]. GUSTO provides two modes of operation. In mode 1, it creates a general-purpose architecture and its datapath for given inputs. In mode 2, it optimizes/customizes the general architecture to improve its area results and design quality. Mode 2 performs this improvement by trimming/removing the unused resources from the general-purpose architecture and creating a scheduled, static, application-specific architecture while ensuring that correctness of the solution is maintained. GUSTO also creates required HDL files which are ready to simulate, synthesize, and map.

The main contributions of this article are:

(1) an easy-to-use matrix inversion core generator for design space exploration with reconfigurable matrix dimensions, bit widths, resource allocation, modes, and methods which can generate and/or optimize the design;

(2) a study of the area, timing, and throughput trade-offs using different design space decisions;

(3) determination of inflection points, in terms of matrix dimensions and bit widths, between QR, LU, and Cholesky decomposition methods and an analytic method.

The rest of this article is organized as follows: Section 2 introduces MIMO systems, matrix inversion, and four methods to solve matrix inversion: QR, LU, and Cholesky decomposition methods and the analytic method. Section 3 explains the architectural design of the core generator, GUSTO. Section 4 describes the error analysis GUSTO uses to determine the accuracy offered by different bit widths. Section 5 introduces FPGA resources, discusses design decisions and challenges, presents implementation results in terms of area and performance, and compares our results with other published FPGA implementations. We conclude in Section 6.

2. MATRIX INVERSION AND ITS METHODS

Explicit matrix inversion of a full matrix is a computationally intensive method. If the inversion is encountered, one should consider converting this problem into an easy decomposition problem which will result in analytic simplicity and computational convenience. Next we describe three known decomposition methods to perform matrix

inversion: QR, LU, and Cholesky decomposition methods [Golub and Loan 1996]. For square matrices, *n* denotes the size of the matrix such that *n* = 4 for 4 x 4 matrices. For rectangular matrices, *m* and *n* denote the number of rows and columns in the matrix, respectively, such that *m* = 3, *n* = 4 for 3 x 4 matrices.

[Insert Figure 1]

QR. Given *A* R^{mxn} with *rank*(A) = *n*, QR factorization exists as *A* = *Q* x *R* where *Q* R^{mxn} has orthonormal columns and *R* R^{nxn} is upper triangular. *LU*. Given *A* R^{nxn} with det(A(1 : *k*, 1 : *k*)) =A 0 for *k* = 1 : *n* − 1, LU decomposition exists as *A* = *L* x *U*. If LU decomposition exists and the given matrix, *A*, is nonsingular, then the decomposition is unique and det(A) = u11 ... u*nn*. *Cholesky*. Given asymmetric positive definite matrix, *A* R^{mxn}, Cholesky decomposition exists as A = G x G^T where *G* R^{mxn} is a unique lower triangular matrix with positive diagonal entries. A matrix *A* E R^{nxn} is *positive definite* if $\boldsymbol{x^T Ax} > \mathbf{0}$ for $\boldsymbol{x}$ E R^n and $\boldsymbol{x}$ =A 0 and if it is *symmetric positive definite matrix* then $\boldsymbol{A^T} = \boldsymbol{A}$. A positive definite matrix is always nonsingular and its determinant is always positive.

Decomposition methods are generally viewed as the preferred methods for matrix inversion because they scale well for large matrix dimensions while the complexity of the analytic method increases dramatically as the matrix dimensions grow. However, for small matrices, the analytic method, which can exploit a significant amount of parallelism, outperforms the decomposition methods. Also note that Cholesky and LU decompositions work only with positive definite and nonsingular diagonally dominant square matrices, respectively. QR decomposition, on the other hand, is more general and can be applied to any matrix. We further explain these different matrix inversion methods, their character-istics and algorithms, the resulting matrices, and the solution steps for matrix inversion in the next subsections.

[Insert Figure 2]

2.1 Matrix Inversion of Triangular Matrices

Triangular matrix inversion is used in all of the decomposition-based (QR, LU, and Cholesky) matrix inversion architectures described before and we use this subsection to describe why this inversion is relatively simple and therefore not a dominant calculation in any of these methods. Primarily, triangular matrix inversion requires fewer calculations

compared to full matrix inversion because of its zero entries. The algorithm for triangular matrix inversion is shown in Figure 1 and described next.

Upper triangular matrix inversion is performed column by column. Calculating the diagonal entries of the R^{-1} matrix consists of simply dividing 1 by the diagonal entry of the R matrix (3) and the rest of the column entries introduce multiplication and addition iteratively (1) which is then divided by the diagonal R matrix entry (2).

2.2 QR Decomposition-Based Matrix Inversion

QR decomposition is an elementary operation which decomposes a matrix into an orthogonal and a triangular matrix. QR decomposition of a matrix A is shown as $A = Q \times R$, where Q is an orthogonal matrix, $Q^T \times Q = Q \times Q^T = I$, $Q-1 = Q^T$, and R is an upper triangular matrix (Figure 2(b)). The solution for the inversion of matrix A, A^{-1}, using QR decomposition is shown as follows.

$$A-1 = R^{-1} \times Q^T \quad (1)$$

This solution consists of three different parts: QR decomposition, matrix inversion for the upper triangular matrix, and matrix multiplication (Figure 2(c)). QR decomposition is the dominant calculation where the next two parts are relatively simple due to the upper triangular structure of R (as described earlier in Section 2.1).

There are three different QR decomposition methods: Gram-Schmidt orthogonormalization (classical or modified), Givens Rotations (GR), and householder reflections. Applying slight modifications to the Classical Gram-Schmidt (CGS) algorithm gives the Modified Gram-Schmidt (MGS) algorithm [Golub and Loan 1996].

QRD-MGS is numerically more accurate and stable than QRD-CGS and it is numerically equivalent to the Givens Rotations solution [Bj″orck et al. 1992, 1994; Singh et al. 2007] (the solution that has been the focus of previously published hardware implementations because of its stability and accuracy). Also, if the input matrix, A, is well-conditioned and nonsingular, the resulting matrices, Q and R, satisfy their required matrix characteristics and QRD-MGS is accurate to floating-point machine precision [Singh et al. 2007]. We therefore present the QRD-MGS algorithm in Figure 2(a) and describe it next.

[Insert Figure 3]

A, Q, R, and X are the input, orthogonal, upper triangular, and intermediate matrices, respectively. The intermediate matrix is the updated input matrix throughout the solution

steps. Matrices with only one index as *Ai* or *Xj* represent the columns of the matrix and matrices with two indices like $_{Rij}$ represent the entry at the intersection of *i*th row with *j*th column of the matrix where $1 < i, j < n$.

In Figure 2(a) we show that we start every decomposition by transferring the input, 4 x 4, matrix columns, *Ai*, into the memory elements (2). Diagonal entries of the *R* matrix are the Euclidean norm of the intermediate matrix columns which is shown as (4). The *Q* matrix columns are calculated by the division of the intermediate matrix columns by the Euclidean norm of the intermediate matrix column, which is the diagonal element of *R* (5). Nondiagonal entries of the *R* matrix are computed by projecting the *Q* matrix columns onto the intermediate matrix columns one by one (7) such that after the solution of *Q*2, it is projected onto *X*3 and *X*4 to compute $_{R23}$ and $_{R24.}$ Lastly, the intermediate matrix columns are updated by (8).

2.3 LU Decomposition-Based Matrix Inversion

If*A* is a square matrix and its leading principal submatrices are all nonsingular, matrix *A* can be decomposed into unique lower triangular and upper triangular matrices. The LU decomposition of a matrix *A* is shown as $A = L \times U$, where *L* and *U* are the lower and upper triangular matrices, respectively (Figure 3(b)).

[Insert Figure 4]

The solution for the inversion of a matrix *A*, A^{-1}, using LU decomposition is shown as follows.

$$A{-}1 = U^{-1} \times L^{-1} \quad (2)$$

This solution consists of four different parts: LU decomposition of the given matrix, matrix inversion for the lower triangular matrix, matrix inversion of the upper triangular matrix, and matrix multiplication (Figure 3(c)). LU decomposition is the dominant calculation where the next three parts are relatively simple due to the triangular structure of the matrices *L* and *U*.

The LU algorithm is shown in Figure 3(a). It writes lower and upper triangular matrices onto the *A* matrix entries. Then it updates the values of the *A* matrix column by column ((4) and (7)). The final values are computed by the division of each column entry by the diagonal entry of that column (9).

2.4 Cholesky Decomposition-Based Matrix Inversion

Cholesky decomposition is another elementary operation which decomposes a symmetric

positive definite matrix into a unique lower triangular matrix with positive diagonal entries. Cholesky decomposition of a matrix$A$ is shown as $A = G \times G^T$, where G is a unique lower triangular matrix, Cholesky triangle, and GT is the transpose of this lower triangular matrix (Figure 4(b)). The solution for the inversion of a matrix, A^{-1}, using Cholesky decomposition is shown as follows.

$$A{-}1 = (G^T)^{-1} \times G^{-1} \quad (3)$$

This solution consists of four different parts: Cholesky decomposition, matrix inversion for the transpose of the lower triangular matrix, matrix inversion of the lower triangular matrix, and matrix multiplication (Figure 4(c)). Cholesky decomposition is the dominant calculation where the next three parts are relatively simple due to the triangular structure of the matrices G and G^T. Figure 4(a) shows the Cholesky decomposition algorithm. We start decomposition by transferring the input matrix, A, into the memory elements. The diagonal entries of lower triangular matrix, G, are the square root of the diagonal entries of the given matrix (2). We calculate the entries below the diagonal entries by dividing the corresponding element of the given matrix by the belonging column diagonal element (4). The algorithm works column by column and after the computation of the first column of the diagonal matrix with the given matrix entries, the elements in the next columns are updated (7). For example, after the computation of $_{G11}$ by (2), $G21$, $G31$, $G41$ by (4), second column: $A22$, $A32$, $A42$, third column: $A33$, $A43$, and fourth column: $_{A44}$ are updated by (7).

2.5 Matrix Inversion Using the Analytic Method

Another method for inverting an input matrix A, is the analytic method which uses the adjoint matrix, $Adj(A)$, and determinant, $det\ A$. This calculation is given by

$$A \quad 1{\sim} \quad 1$$

$$\det A \times Adj(A). \quad (4)$$

The adjoint matrix is the transpose of the cofactor matrix where the cofactor matrix is formed by using determinants of the input matrix with signs depending on its position. It is formed in three stages. First, we find the transpose of the input matrix, A, by interchanging the rows with the columns. Next, the matrix of minors is formed by covering up the elements in its row and column and finding the determinant of the remaining matrix. Finally, the cofactor of any element is found by placing a sign in front of the matrix of minors by calculating $(-1)^{(i+j)}$. These calculations are shown in Figure 5(a) for the first

entry in the cofactor matrix, $C11$.

The calculation of the first entry in the cofactor matrix $_{C11}$ is also presented in Figure 5(b) using a cofactor calculation core. This core is run 16 times for a 4 x 4 matrix to form the 4 x 4 cofactor matrix which has 16 entries. The adjoint matrix is the transpose of the cofactor matrix and formed using register renaming. After the calculation of the adjoint matrix, the determinant is calculated using a row or a column which is shown in (c) using the determinant calculation core. The last stage is the division between the adjoint matrix and the determinant which gives the inverted matrix.

For the analytic method, we present three different designs, *Implementation A, B,* and *C,* with varying levels of parallelism (using cofactor calculation cores in parallel) to form cofactor matrices. *Implementation A* uses one cofactor calculation core, *implementation B* uses two cofactor calculation cores, and *implementation C* uses 4 cofactor calculation cores. In the next section, we present our core generator GUSTO which is an infrastructure for fast prototyping the matrix inversion architectures using different methods.

[Insert Figure 5]

3. MATRIX INVERSION CORE GENERATOR TOOL

There are several different architectural design alternatives for these solution methods of matrix inversion. Thus, it is important to study trade-offs between these alternatives and find the most suitable solution for desired results such as the most time efficient or most area efficient design. Performing design space exploration is a time-consuming process where there is an increasing demand for higher productivity. High-level design tools offer great convenience by easing this burden and giving us the opportunity to test different alternatives in a reasonable amount of time. Therefore, designing a high-level tool for fast prototyping is essential.

GUSTO (**G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization) is such a high-level design tool, written in Matlab, that is the first of its kind to provide design space exploration across different matrix inversion architectures. As shown in Figure 6, GUSTO allows the user to select the matrix inversion method (QR, LU, Cholesky decompositions, or analytic), the matrix dimension, the type and number of arithmetic resources, the data representation (the integer and fractional bit width), and the mode of operation (mode 1 or mode 2).

Mode 1 of GUSTO generates a general-purpose architecture and its datapath by using resource-constrained list scheduling after the required inputs are given. The general-purpose architecture is used for area and timing analysis for a general nonoptimized solution. The advantage of generating a general-purpose architecture is that it can be used to explore other algorithms, so long as these algorithms require the same resource library. However, mode 1's general-purpose architectures generally do not lead to high-performance results. Therefore optimizing/customizing these architectures to improve their area results is another essential step to enhance design quality.

[Insert Figure 6]

GUSTO creates a CPU-like architecture which can be seen in Figure 7. The created architecture works at the instruction level where the instructions define the required calculations for the matrix inversion. For better performance results, instruction-level parallelism is exploited. The dependencies between the instructions limit the amount of parallelism that exists within a group of computations. Our proposed design consists of controller units and arithmetic units. The arithmetic units are capable of computing decomposition, simple matrix inversion using back-substitution, and matrix multiplication by employing adders, subtractors, multipliers, dividers, and square root units that are needed. In this architecture, controller units track the operands to determine whether they are available and assign a free arithmetic unit for the desired calculation. Every arithmetic unit fetches and buffers an operand as soon as the operand is ready.

In mode 2, GUSTO performs this improvement by trimming/removing the unused resources from the general-purpose architecture and creating a scheduled, static, application-specific architecture while ensuring that correctness of the solution is maintained. GUSTO simulates the architecture to define the usage of arithmetic units, multiplexers, register entries, and input/output ports and trims away the unused components with their interconnects.

A trimming example is shown in Figure 8. Suppose there are 2 arithmetic units with 2 inputs/1 output each and one memory with 1 input/2 outputs (Figure 8(a)). Input/output port relationships between arithmetic unit *A* and the other units are shown in a block diagram in (Figure 8(b)). Although *Out A, Out B, Out mem1,* and *Out mem2* are all inputs to *In A1* and *In A2,* but not all the inputs may be used during computation. We can

represent whether an input/output port is used or not during simulation in a matrix such as the one shown in (Figure 8(c)). As the simulation runs, the matrix is filled with 1s and 0s representing the used and unused ports, respectively. GUSTO uses these matrices to remove the unused resources (Figure 8(d)). In this example, two inputs, *Out A*, *Out mem1* to *In A1* and another two inputs, *Out B*, *Out mem2* to *In A2* are removed.

[Insert Figure 7 &8]

4. FIXED-POINT ARITHMETIC AND ERROR ANALYSIS USING GUSTO

There are two different types of approximations for real numbers: fixed-point and floating-point arithmetic systems. Floating-point arithmetic represents a large range of numbers with some constant relative accuracy. Fixed-point arithmetic represents a reduced range of numbers with a constant absolute accuracy. Usage of floating-point arithmetic is expensive in terms of hardware and leads to inefficient designs, especially for FPGA implementation. On the other hand, fixed-point arithmetic results in efficient hardware designs with the possibility of introducing calculation error.

We use two's complement fixed-point arithmetic in our implementations as it results in faster and smaller functional units. The data lines used in our implementations for fixed-point arithmetic consist of an integer part, a fractional part, and a sign bit. Fixed-point arithmetic reduces accuracy and consequently introduces two types of errors: round-off and truncation errors. Round-off error occurs when the result requires more bits than the reserved bit width after a computation. Truncation error occurs due to the limited number of bits to represent numbers. These issues must be handled carefully to prevent incorrect or low-accuracy results. Thus, error analysis is a crucial step to determine how many bits are required to satisfy accuracy requirements.

GUSTO performs error analysis after the instruction generation step (shown in Figure 6) to find an appropriate fixed-point representation which provides results with accuracy similar to that of a floating-point implementation. GUSTO takes the sample input data which is generated by the user. The matrix inversion is performed using single or double precision floating-point arithmetic and these are referred as the actual results. The same calculations are performed using different bit widths of fixed-point representations to determine the error, the difference between the actual and the computed result. GUSTO provides four

different metrics to the user to determine if the accuracy is enough for the application: mean error, standard deviation of error, peak error, and mean percentage error, as shown in Figure 9.

The first metric, mean error, is computed by finding the error for all matrix entries and then dividing the sum of these errors by the total number of entries. This calculation can be seen as where y, $\hat{y}$, and m are the actual results, the computed results, and the number of entries which are used in the decomposition (16 for a 4 × 4 matrix), respectively. Mean error is an important metric for error analysis, however, it

$$\frac{\sum_{i=1}^{m} |y_i - y_i|}{m}, \tag{5}$$

[Insert Figure 9]

does not include the information about outlier errors. This is the case where a small number of entries have very high error but the majority of entries have very small error. To calculate the dispersion from the mean error, the standard deviation of error and the peak error are introduced in our tool. Mean error sometimes leads to misleading conclusions if the range of the input data is small. Therefore the third metric, mean percentage error, makes more sense if the relative error is considered. This metric is defined as

$$\frac{\sum_{i=1}^{m} \left| \frac{y_i - y_i}{y_i} \right|}{m}. \tag{6}$$

As an example, we perform an error analysis for QR decomposition-based matrix inversion. We generate uniformly distributed pseudorandom numbers, [0, 1], for a 4 × 4 matrix. The mean error results provided by GUSTO are shown in Figure 10 in log domain where mean error decreases with the increase in the number of bits used as bit width. Therefore, the user can determine how many bits are required for the desired accuracy. It is important to note that the tool also provides standard deviation of error, peak error, and mean percentage error.

5. RESULTS

In this section, we present different design space exploration examples using different inputs of GUSTO and compare our results with previously published FPGA implementations. Design space exploration can be divided into two parts: inflection point analysis and architectural design alternatives analysis.

Inflection Point Analysis. In this subsection, we first compare QR decomposition and analytic method because they are both applicable to any matrix. Then, we compare different decomposition methods (QR, LU, and Cholesky) to benefit from different matrix characteristics.

[Insert Figures 10 & 11]

Comparison of QR Decomposition-Based Matrix Inversion and Analytic Method. The total number of operations used in these methods is shown in Figure 11 in log domain. It is important to notice that the total number of operations increases by an order of magnitude for each increase in matrix dimension for the analytic method, making the analytic solution unreasonable for large matrix dimensions. Since the analytic approach does not scale well, there will be an inflection point where the QR decomposition approach will provide better results. At what matrix size does this inflection point occur and how does varying bit width and degree of parallelism change the inflection point? The comparisons for sequential and parallel executions of QR and analytic methods are shown in Figures 12 and 13 with different bit widths: 16, 32, and 64. We used implementation A for the parallel implementation of the analytic method. Solid and dashed lines represent the QR decomposition method and analytic method results, respectively. The balloons denote the inflection points between the two methods for the different bit widths.

[Insert Figures 12 & 13]

The sequential execution results (Figure 12) show that the analytic method offers a practical solution for matrix dimensions < 4 x 4. It also gives the same performance as the QR decomposition method for 5 x 5 matrices using 64 bits. The analytic method result increases dramatically for 6 x 6 matrices (not shown) where it needs 12,251 clock cycles (for 16 bits) as opposed to 1,880 clock cycles for QR decomposition, suggesting the analytic method is unsuitable for matrix dimensions >6 x 6.

The parallel execution results are shown in Figure 13. The analytic method offers a practical solution for matrix dimensions <4 x 4 and it is preferred for 5 x 5 matrix dimension for 32 and 64 bits. The increase in the clock cycle is again dramatic for matrix dimensions >6 x 6 for the

analytic method. This requires to use the QR decomposition method for these larger matrix dimensions.

[Insert Figure 14]

Comparison of Different Decomposition Methods. The total number of operations used in different decomposition-based matrix inversion architectures is shown in Figure 14 in log domain. It is important to notice that there is an inflection point between LU and Cholesky decompositions at 4 x 4 matrices with a significant difference from QR decomposition. The comparisons for sequential and parallel executions of QR, LU, and Cholesky, decomposition-based matrix inversion architectures are shown in Figures 15 and 16, respectively, with different bit widths: 16, 32, and 64. Square, spade, and triangle represent QR, LU, and Cholesky, methods, respectively. Solid, dashed, and smaller dashed lines represent 64, 32, and 16 bits of bit widths, respectively. The balloons denote the inflection points between these methods for the different bit widths where an inflection point occurs. The sequential execution results of decomposition-based matrix inversion architectures (Figure 15) show that QR takes more clock cycles than Cholesky and LU, where Cholesky takes more cycles than LU. As the bit widths get smaller, the difference between QR and the others doesn't change significantly, however, it becomes smaller between Cholesky and LU decomposition-based inversions. There is an inflection point between LU and Cholesky decompositions at 7 x 7 matrices for 16 bits. The parallel execution results of decomposition-based matrix inversion (Figure 16) show that QR decomposition-based matrix inversion architectures have the highest number of clock cycles for all bit widths. Cholesky and LU decomposition-based matrix inversion architectures have a similar number of clock cycles for small bit widths. However, LU decomposition uses increasingly fewer clock cycles than Cholesky decomposition with increasing bit widths and matrix dimensions. LU decomposition with 32 bits performs almost the same as QR decomposition with 16 bits. Also, 64-bits LU decomposition performs almost the same as 32-bits QR decomposition in terms of total number of clock cycles.

[Insert Figures 15 &16]

Architectural Design Alternatives. These analyses are shown for QR, LU, and Cholesky, decomposition-based matrix inversion architectures for 4 × 4 matrices. We present area

results in terms of slices and performance results in terms of throughput. Throughput is calculated by dividing the maximum clock frequency (MHz) by the number of clock cycles to perform matrix inversion. All designs are written in Verilog and synthesized using Xilinx ISE 9.2. Resource utilization and design frequency are post place-and-route values obtained using a Virtex 4 SX35 FPGA.

[Insert Figure 17]

All functional units are implemented using the Xilinx Coregen toolset. The addition and subtraction units are implemented with SLICES, the multiplications use XtremeDSP blocks, the divider core uses a circuit for fixed-point division based on radix-2 nonrestoring division, and the square root unit uses a CORDIC core. We use Block RAMs available on Xilinx FPGAs as memory storage space for instructions. The Block RAM modules provide flexible 18Kbit dual-port RAM, that are cascadable to form larger memory blocks. Embedded XtremeDSP SLICES with 18 × 18 bit dedicated multipliers and a 48-bit accumulator provide flexible resources to implement multipliers to achieve high performance. Furthermore, the Xilinx Coregen tool set implements these cores very efficiently since it uses special mapping and place-and-route algorithms allowing for high-performance design.

We present both mode 1 (nonoptimized) and mode 2 (optimized) results in Figure 17 to show the improvement in our results with the optimization feature, and present only mode 2 results in Figures 18 and 19.

We investigate different resource allocations for QR decomposition-based matrix inversion architectures using both modes of GUSTO and present the results in Figure 17. As expected from mode 1, Figure 17 shows an increase in area and throughput as the number of resources increase up to the optimal number of resources. Adding more than the optimal number of resources decreases throughput while still increasing area. However, mode 2 of GUSTO finds the optimal number of resources, which maximizes the throughput while minimizing area (shown in Figure 17). Mode 2’s optimized application-specific architecture can therefore provide an average 59% decrease in area and 3X increase in throughput over mode 1’s general-purpose (nonoptimized) design.

Bit width of the data is another important input for the matrix inversion. The precision of the results is directly dependent on the number of bits used. The usage of a high number of

bits results in high precision at a cost of higher area and lower throughput. We present 3 different bit widths, 19, 26, and 32 bits, in Figure 18 for these three different decomposition-based matrix inversion architectures. Usage of LU decomposition for matrix inversion results in the smallest area and highest throughput compared to the other methods. Cholesky decomposition offers higher throughput at a cost of larger area compared to QR decomposition.

[Insert Figures 18 &19]

We also present three different matrix dimensions, 4x4, 6x6, and 8x8, with implementation results in Figure 19 showing how the area and performance results scale with matrix dimension. We again observe that LU decomposition-based matrix inversion architectures offer better area and throughput results compared to other methods.

Comparison. A comparison between our results and previously published implementations for a 4 x 4 matrix is presented in Tables I and II. For ease of comparison we present all of our implementations with bit width 20, as this is

[Insert Tables I &II]

the largest bit width value used in the related works. Though it is difficult to make direct comparisons between our designs and those of the related works (because we used fixed-point arithmetic instead of floating-point arithmetic and fully used FPGA resources (like DSP48s) instead of LUTs), we observe that our results are comparable. The main advantages of our implementation are that it provides the designer the ability to study the trade-offs between architectures with different design parameters and provides a means to find an optimal design.

6. CONCLUSION

This article describes a matrix inversion core generator tool, GUSTO, that we developed to enable easy design space exploration for various matrix inversion architectures which targets reconfigurable hardware designs. GUSTO provides different parameterization options, including matrix dimensions, bit widths, and resource allocations, which enables us to study area and performance tradeoffs over a large number of different architectures. We present QR, LU, and Cholesky decomposition methods and an analytic method for matrix inversion, to observe the

advantages and disadvantages of all of these methods in response to varying parameters. GUSTO is the only tool that allows design space exploration across different matrix inversion architectures. Its ability to provide design space exploration, which leads to an optimized architecture, makes GUSTO an extremely useful tool for applications requiring matrix inversion (i.e., MIMO systems).

REFERENCES

ABE, T., TOMISATO, S., AND MATSUMOTO, T. 2003a. A MIMO turbo equalizer for frequency-selective channels with unknown interference. *IEEE Trans. Vehicular Technol. 52*, 3, 476–482.

ABE, T. AND MATSUMOTO, T. 2003b. Space-Time turbo equalization in frequency selective MIMO channels. *IEEE Trans. Vehicular Technol.* 469–475.

BJ″ORCK, A. AND PAIGE, C. 1992. Loss and recapture of orthogonality in the modified Gram-Schmidt algorithm. *SIAM J. Matrix Anal. Appl. 13*, 1, 176–190.

BJ″ORCK, A. 1994. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl. 198*, 297– 316.

CAGLEY, R. E., WEALS, B. T., MCNALLY, S. A., ILTIS, R. A., MIRZAEI, S., AND KASTNER, R. 2007. Implementation of the Alamouti OSTBC to a distributed set of single-antenna wireless nodes. In *Proceedings of the IEEE Wireless Communications and Networking Conference. IEEE*. 577–581.

EDMAN, F. AND ¨OWALL, V. 2005. A scalable pipelined complex valued matrix inversion architecture. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* 4489–4492.

EILERT, J., WU, D., AND LIU, D. 2007. Efficient complex matrix inversion for MIMO software defined radio. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* 2610–2613.

GOLUB, G. H. AND LOAN, C. F. V. 1996. *Matrix Computations* 3rd Ed. John Hopkins University Press, Baltimore, MD.

HANGJUN, C., XINMIN, D., AND HAIMOVICH, A. 2003. Layered turbo space-time coded MIMO-OFDM systems for time varying channels. In *Proceedings of the IEEE Global Telecommunications Conference, 4*, 1831–1836.

IEEE 802.11. LAN/MAN wireless LANS. IEEE Standards Association. http://standards.ieee.org/

getieee802/802.11.html
IEEE 802.16. LAN/MAN broadband wireless LANS. IEEE Standards Association. http://standards. ieee.org/getieee802/802.16.html
ILTIS, R. A., MIRZAEI, S., KASTNER, R., CAGLEY, R. E., AND WEALS, B. T. 2006. Carrier offset and channel estimation for cooperative MIMO sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*. 1–5.
IRTURK, A., BENSON, B., MIRZAEI, S., AND KASTNER, R. 2008. An FPGA design space exploration tool for matrix inversion architectures. In *Proceedings of the IEEE Symposium on Application Specific Processors (SASP).*
KARKOOTI, M., CAVALLARO, J. R., AND DICK, C. 2005. FPGA implementation of matrix inversion using QRD-RLS algorithm. In *Proceedings of the Conference Record of the 39th Asilomar Conference on Signals, Systems and Computers*. 1625–1629.
KUSUME, K., JOHAM, M., UTSCHICK, W., AND BAUCH, G. 2005. Efficient Tomlinson-Harashima pre-coding for spatial multiplexing on flat MIMO channel. In *Proceedings of the IEEE International Conference on Communications*, vol. 3. 2021–2025.
MENG, Y., BROWN, A. P., ILTIS, R. A., SHERWOOD, T., LEE, H., AND KASTNER, R. 2005. MP core: Algorithm and design techniques for efficient channel estimation in wireless applications. In *Proceedings of the 42nd Design Automation Conference*. 297–302.
SINGH, C. K., PRASAD, S. H., AND BALSARA, P. T. 2007. VLSI architecture for matrix inversion using modified Gram-Schmidt based QR decomposition. In *Proceedings of the 20th International Conference on VLSI Design*. 836–841.
ZHOU, L., QIU, L., AND ZHU, J. 2005. A novel adaptive equalization algorithm for MIMO communication system. In *Proceedings of the Vehicular Technology Conference*. 2408–2412.

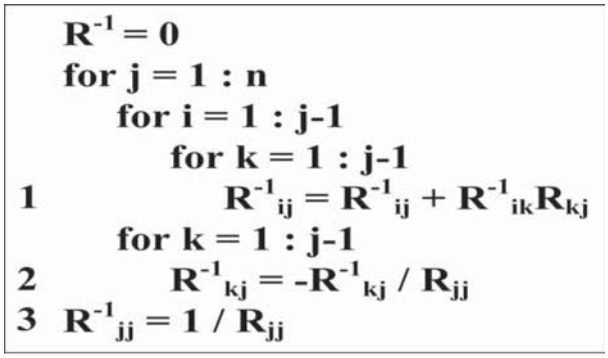
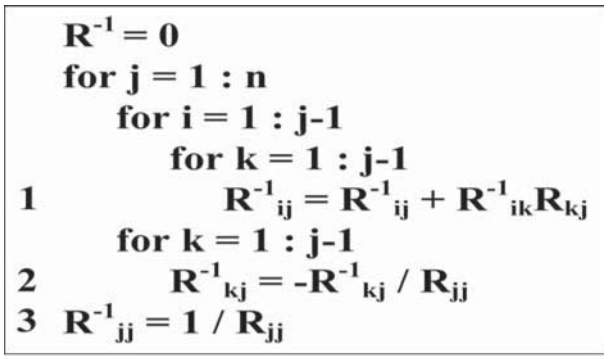


Fig. 1. Matrix inversion of upper triangular matrices.



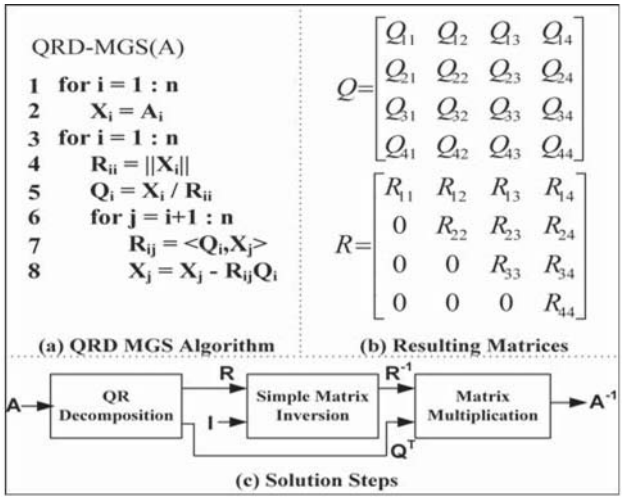


Fig. 2. QR decomposition Modified Gram Schmidt (QR-MGS) algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b). The solution steps of the matrix inversion are presented in (c).

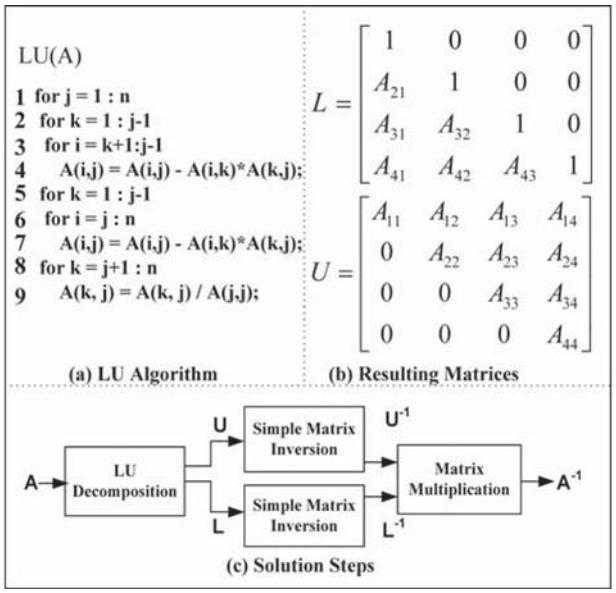


Fig. 3. LU decomposition algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b). The solution steps of the matrix inversion are presented in (c).

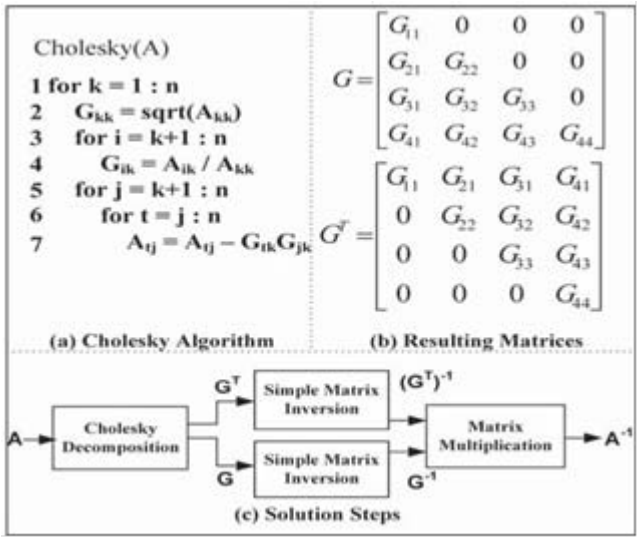


Fig. 4. Cholesky decomposition algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b). The solution steps of the matrix inversion are presented in (c).

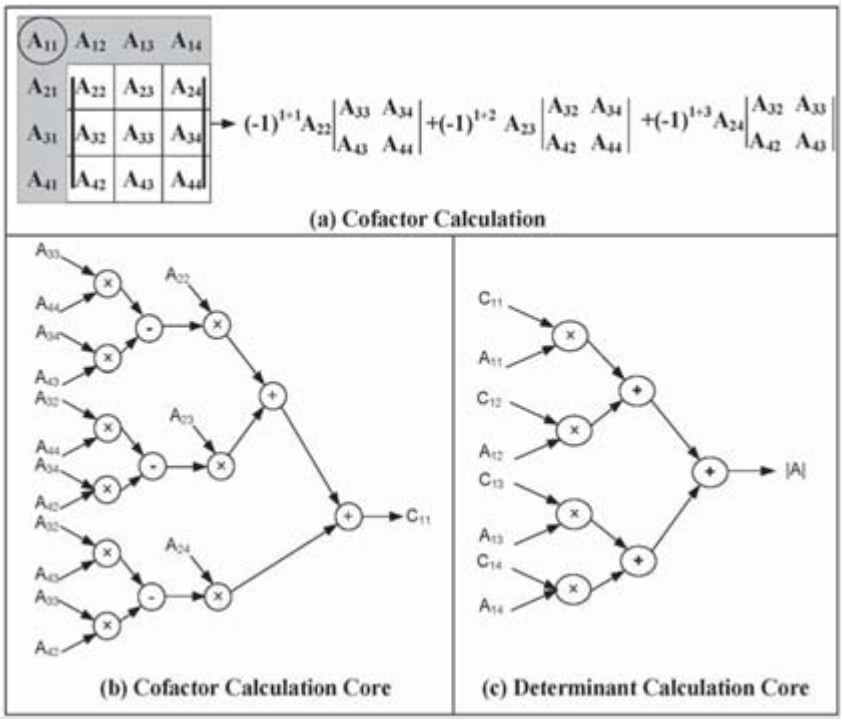


Fig. 5. Matrix Inversion with analytic approach. The first element of cofactor matrix, $_{C11,}$ and determinant calculation for a 4 × 4 matrix is shown in (a), (b) and (c) respectively.

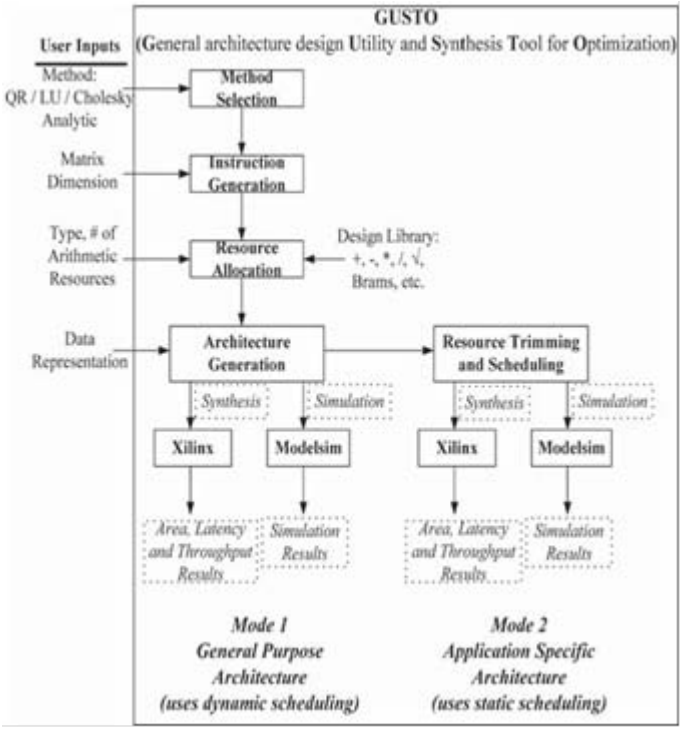
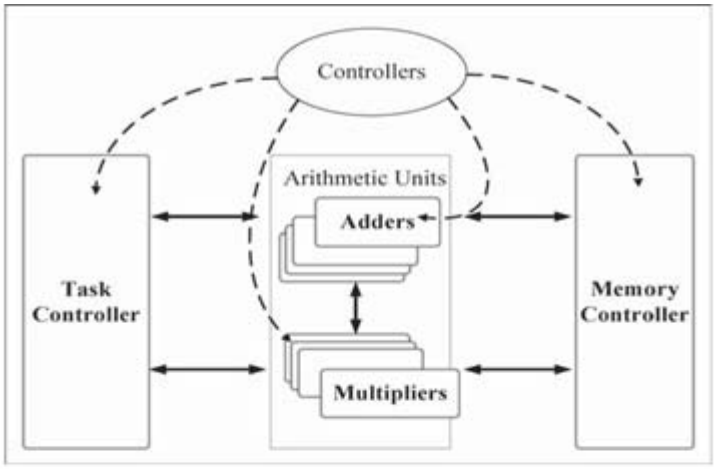


Fig. 6. Different modes of GUSTO.



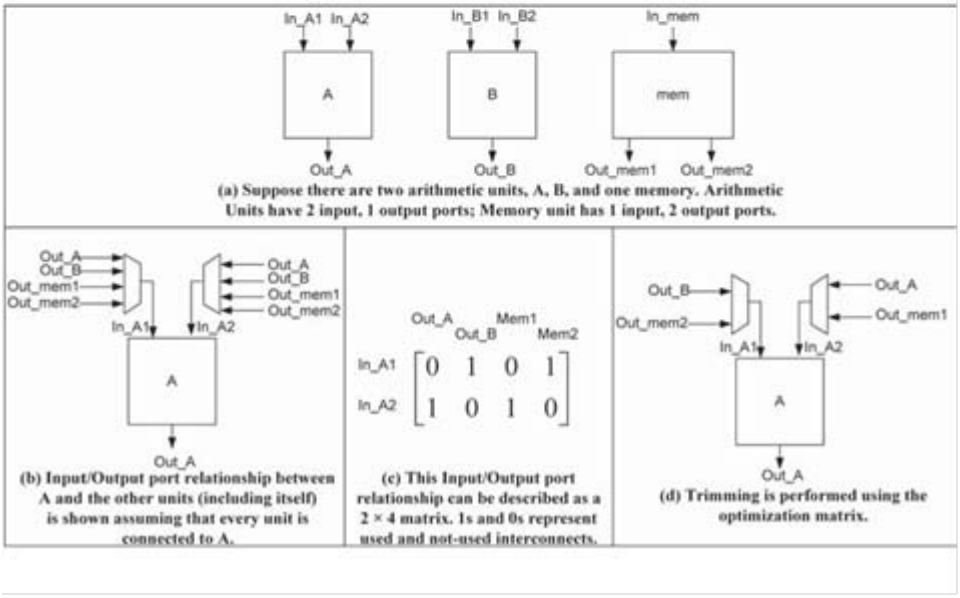


Fig. 7. General-purpose architecture and its datapath.

Fig. 8. Flow of GUSTO's trimming feature.

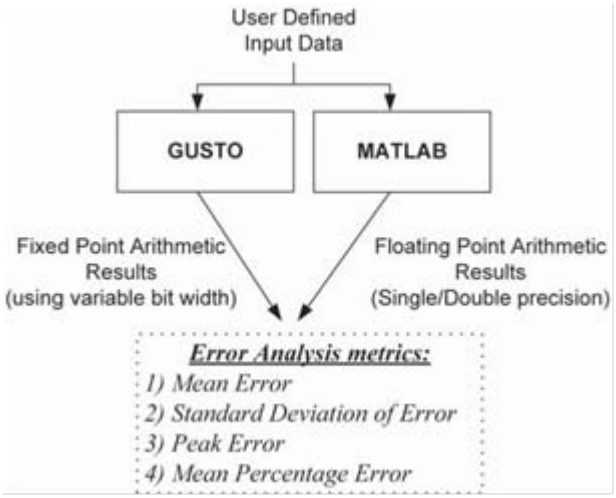


Fig. 9. Performing error analysis using GUSTO.

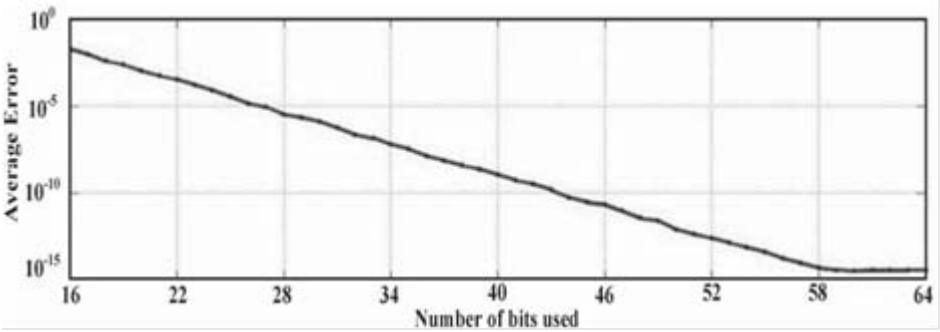


Fig. 10. An error analysis example, mean error, provided by GUSTO for QR decomposition-based 4 × 4 matrix inversion. The user can select the required number of bit widths for the application where the increasing number of bits results in high accuracy.

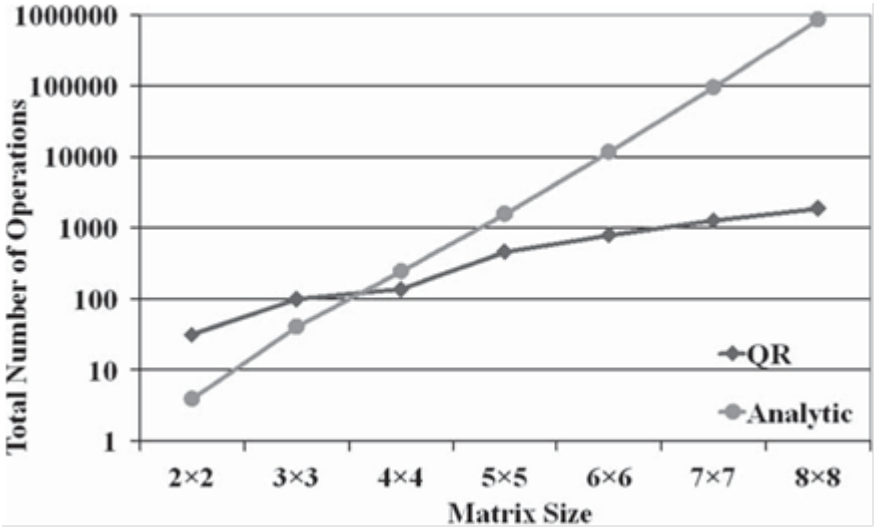


Fig. 11. The total number of operations for both the QR decomposition and the analytic method in log domain.

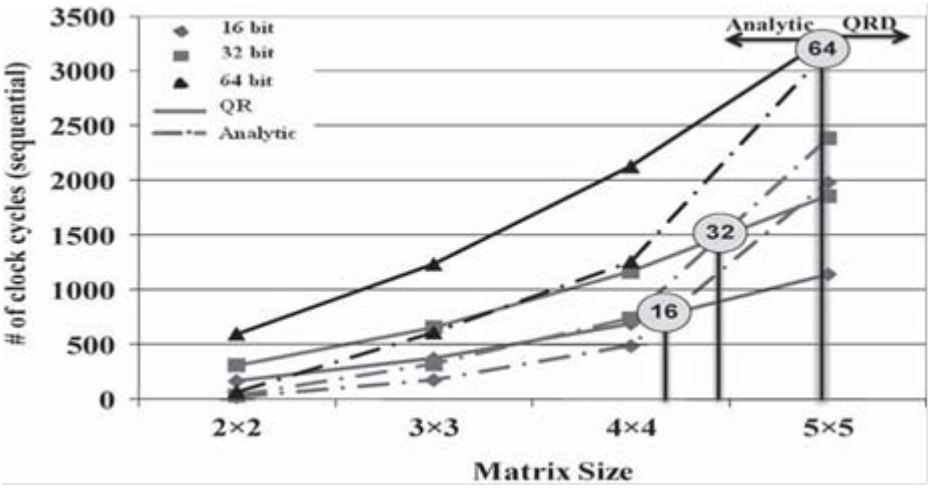


Fig. 12. The inflection point determination between the QR decomposition and the analytic method using sequential execution.

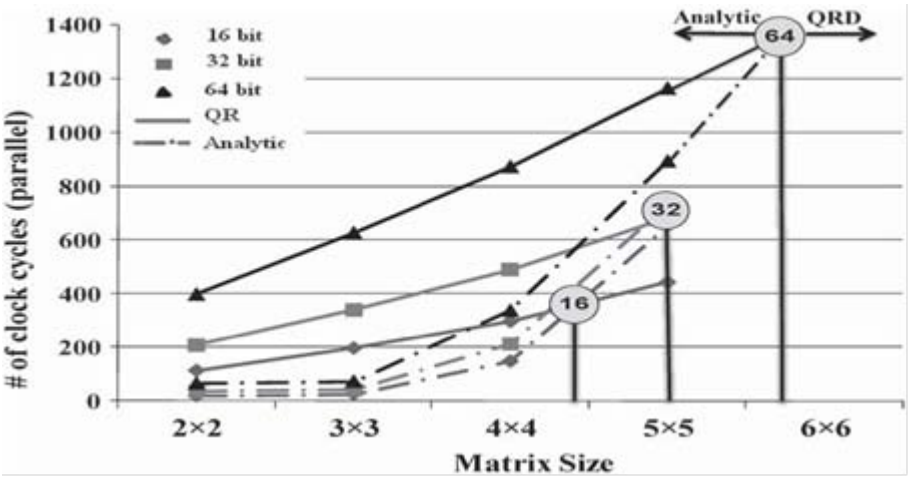


Fig. 13. The inflection point determination between QR decomposition and analytic method using parallel execution.

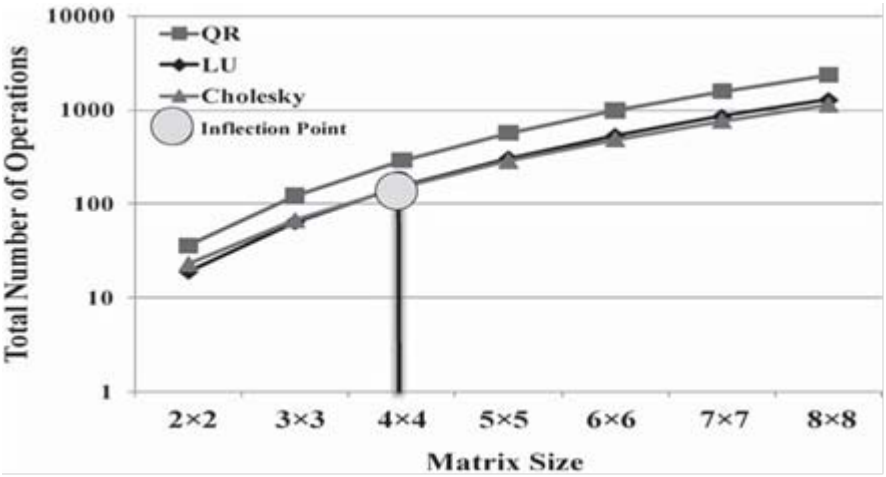


Fig. 14. The total number of operations for different decomposition-based matrix inversion methods in log

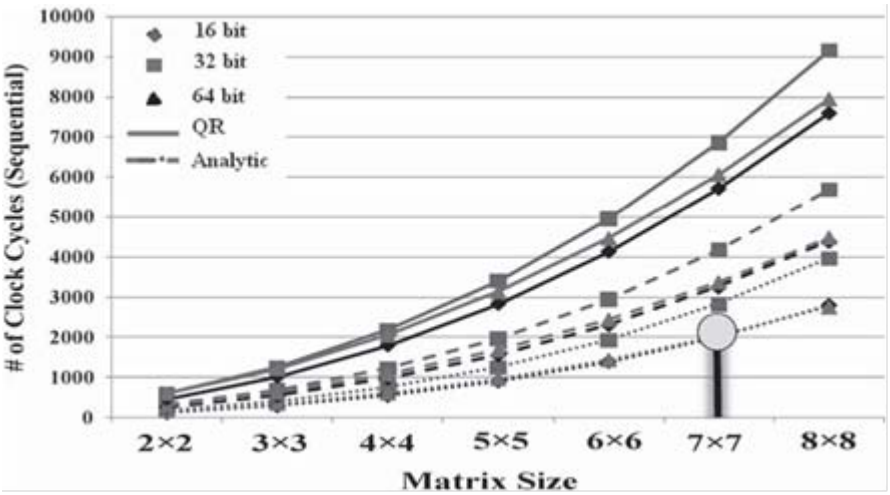


Fig. 15. The comparison between different decomposition-based matrix inversion methods using sequential execution.

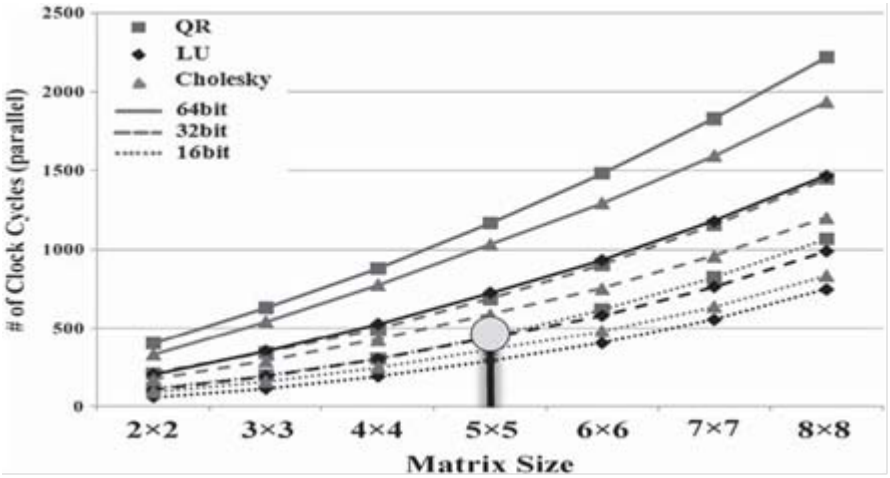


Fig. 16. The comparison between different decomposition-based matrix inversion methods using parallel execution.

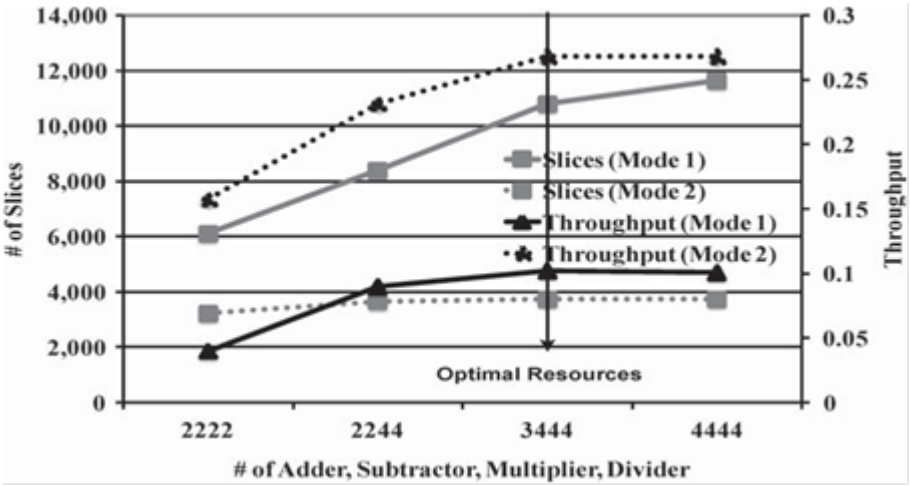


Fig. 17. Design space exploration using different resource allocation options.

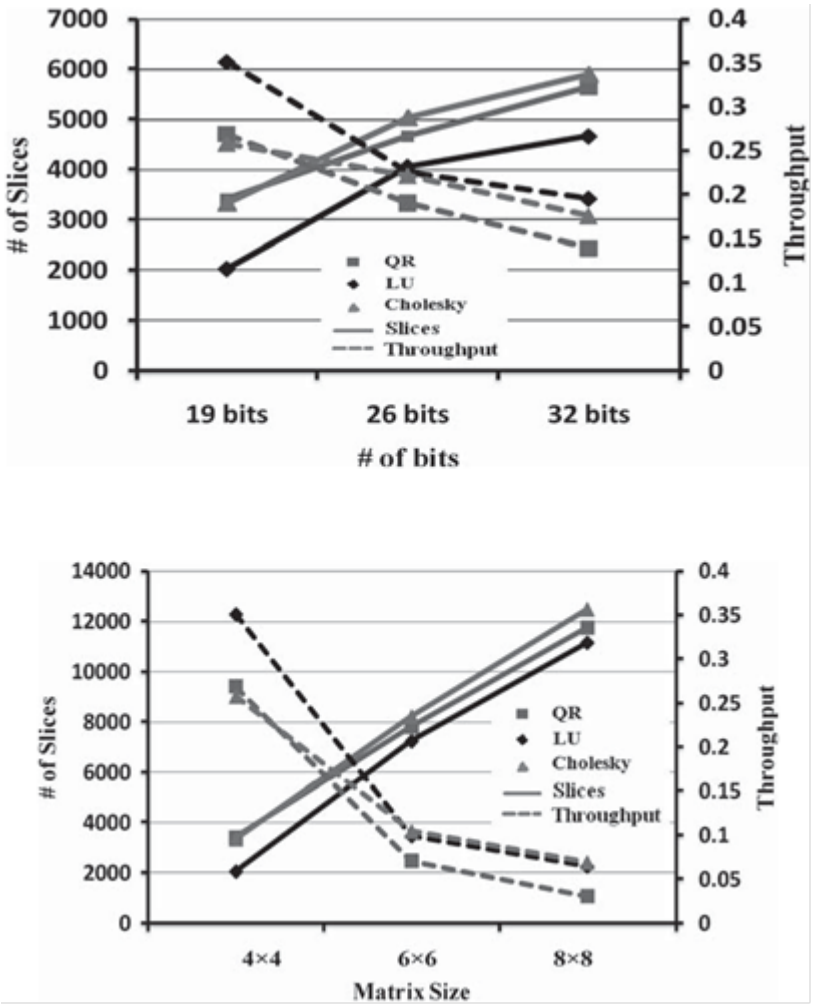


Fig. 18. Design space exploration using different bit widths.

Fig. 19. Design space exploration using different matrix sizes.

Method	*[Eilert et al. 2007]* Anlytic	*[Eilert et al. 2007]* Analytic	*GUSTO* Analytic		
			ImplA	*ImplB*	*ImplC*
Bit width	16	20	20	20	20
Data type	floating	floating	fixed	fixed	fixed
Device type	Virtex 4	Virtex 4	Virtex 4	Virtex 4	Virtex 4
Slices	1561	2094	702	1400	2808
DSP48s	0	0	4	8	16
BRAMs	NR	NR	0	0	0
Throughput $(10^6 \times s^{-1})$	1.04	0.83	0.38	0.72	1.3

NR denotes not reported.

Table I. Comparisons Between Our Results and Previously Published Articles for Analytic

Method	*[Edman et al. 2005]* QR	*[Karkooti et al. 2005]* QR	*GUSTO* QR	*GUSTO* LU	*GUSTO* Cholesky
Bit width	12	20	20	20	20
Data type	fixed	floating	fixed	fixed	fixed
Device type	Virtex 2	Virtex 4	Virtex 4	Virtex 4	Virtex 4
Slices	4400	9117	3584	2719	3682
DSP48s	NR	22	12	12	12
BRAMs	NR	NR	1	1	1
Throughput $(10^6 \times s^{-1})$	0.28	0.12	0.26	0.33	0.25

NR denotes not reported.

Table II. Comparisons Between Our Results and Previously Published Articles for Decomposition Methods.