FPGA Implementation of Adaptive Weight Calculation Core Using QRD-RLS Algorithm

Ali Irturk[†], Shahnam Mirzaei[‡], Ryan Kastner[†]

†Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
{airturk, b1benson, kastner}@cs.ucsd.edu

‡Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106
shahnam@umail.ucsb.edu

Abstract—**We present a novel architecture for adaptive weight calculation (AWC) that uses the QR decomposition based recursive least squares (RLS) algorithm. Our AWC core achieves a throughput of 0.20M updates per second for a 4 x 4 matrix on a Xilinx Virtex4 SX FPGA. We show that our core is significantly faster than other published FPGA implementations and it requires fewer resources. This is largely a consequence of careful error analysis that allows us to take advantage of a fixed point data representation with little degradation in the final results. Finally, our proposed architecture scales well for larger size matrices.**

***Index Terms*—Adaptive Weight Calculation, Field Programmable Gate Arrays, Matrix Decomposition**

I. Introduction

Adaptive weight calculation (AWC) is required in many communication applications including adaptive beamforming, equalization, predistortion and multiple-input multiple-output (MIMO) systems [1]. These applications require solving over-determined systems of equations in many cases. In general, the least squares approach, e.g. Least Mean Squares (LMS), Normalized LMS (NLMS) and Recursive Least Squares (RLS), is used to find an approximate solution to these kinds of system of equations. Among them, RLS is most commonly used due to its good numerical properties and fast convergence rate [2]. However, it requires matrix inversion which is not efficient in terms of precision and hardware implementation. Applying QR decomposition (QRD) to perform adaptive weight calculation based on RLS avoids this problem and leads to more accurate results and efficient architectures.

There are three different QR decomposition methods: Gram-Schmidt orthogonormalization, Givens Rotations (GR) and Householder reflections [3]. GR is preferred because of its stability and accuracy. GR lends itself easily to a systolic array architecture using CORDIC blocks [4] which makes an efficient hardware implementation. Therefore, it is often used for hardware implementation. However, it was shown that the modified Gram-Schmidt (MGS) method is numerically equivalent to Givens rotations method [5]. And our results using QRD-MGS are better than previously published results using GR.

A wide variety of computationally intensive applications are moving from Digital Signal Processors (DSPs) to Field Programmable Gate Arrays (FPGAs) because FPGA architectures present designers with substantially more parallelism allowing more efficient application implementations. Moreover, FPGAs are a flexible, cost effective alternative to Application Specific Integrated Circuits (ASICs).

FPGAs are perfect platforms for arithmetic operations such as matrix decomposition as they provide powerful computational architectural features, e.g. embedded multipliers, shift register LUTs (SRLs), Block RAMs (BRAMs), DSP blocks and DCMs (Digital Clock Managers). If used correctly, these features can enhance the performance and throughput significantly. We will discuss the design decisions that we encountered as we customized our design to utilize the FPGA architectural features.

In this paper we present an architectural scheduling approach for a scalable adaptive weight calculation (AWC) core which uses QRD-RLS with fixed-point arithmetic and map it onto Xilinx Virtex 4SX FPGA. We explore practical hardware design and implementation issues for FPGAs. Our core results in a high performance, scalable architecture for adaptive weight calculation. The remainder of the paper is organized as follows: Section II presents related work. Section III discusses RLS approach and QRD-MGS algorithm for weight calculation. Section IV addresses the advantages of fixed-point arithmetic and subsequently presents error analysis for different amounts of precision. Section V explains the architectural design of AWC core. Section VI introduces FPGA resources, discusses design decision and challenges, and presents implementation results in terms of area and timing. We conclude in Section VII.

II. Related Work

In the most recent work using Modified Gram Schmidt (MGS) algorithm for QR decomposition, Singh et al. implemented a fully parallel VLSI architecture using fixed point arithmetic for matrix inversion [6]. The main contribution of the paper is introducing a Look Up Table (LUT) based approach to the MGS. Lightbody et al. presented generic mapping of a triangular QR architecture for linear or rectangular array of processors using square GR [7]. Other different QR array architectures for adaptive beamforming are presented in [8-10]. However, FPGA implementations of these designs are not considered.

Karkooti et al. presented an FPGA implementation of matrix inversion using the QRD-RLS algorithm along with square GR and folded systolic arrays [11]. Boppana et al. presented a weight calculation core using QRD-RLS [12] which is very similar to our work; however the solution of QR decomposition method and architectural design are different. Dick et al. considered the architecture, design flow and the verification process of a real-time beamformer using FPGAs [13]. The proposed design uses a mixture of CORDIC-based processing and MAC-based arithmetic. Our results are better than these previous solutions as we show in Section VI.

III. RLS Approach and QRD-MGS Algorithm

A. *Recursive Least Squares Approach*

Many communication applications must solve systems of equations for weight calculation which can be seen as

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1 + e_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2 + e_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n + e_n \end{array} \tag{1}$$

A is the observations matrix which is assumed to be noisy, *b* is known training sequence and *x* is the vector to be computed by using least squares method. This is described more compactly in matrix notation:

$$Ax = b + e \tag{2}$$

If there are the same numbers of equations as there are unknowns, i.e. $n = m$, this system of equations has a unique solution, $x = A^{-1}b$. However, the high sampling rate communication applications are often over-determined, i.e. $n > m$. Introducing the least squares approach helps to solve the problem by minimizing the residuals:

$$\min \sum_n e_n{}^2 \tag{3}$$

However RLS approach requires matrix inversion which is expensive for hardware implementation and introduces low precision results due to its complexity. Furthermore the resulting normal equations in the RLS approach are more ill conditioned than the given over-determined system which further affects the precision. For these reasons, QRD–RLS is a better method to solve the problem.

QR decomposition is an elementary operation, which decomposes a matrix into an orthogonal and a triangular matrix. QR decomposition of a matrix *A* is a decomposition of *A as* $A = Q \times R$, where *Q* is an orthogonal matrix ($Q^T \times Q = I$) and *R* is an upper triangular matrix. The decomposition of $m \times n$ matrices (with $m \geq n$) of full rank is the product of an $m \times n$ orthogonal matrix where $Q^T \times Q = I$ and an $n \times n$ upper triangular matrix.

The resulting upper triangular matrix, *R*, which is the solution of QR decomposition, is used to find coefficients of the system by back-substitution after converting *b* into another column matrix, *c*, such that $Rx = c$. The required calculations for AWC are shown in Figure 1.

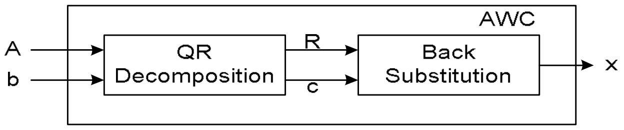


Fig. 1. AWC using QRD-RLS method consists of two different parts to calculate the weights, QR decomposition and back-substitution.

B. *QR decomposition using Modified Gram-Schmidt Algorithm and back-substitution*

Applying slight modifications to the Classical Gram-Schmidt (CGS) algorithm gives the modified Gram-Schmidt (MGS) algorithm. QRD-MGS is numerically more accurate and stable than the QRD-CGS. And it is numerically equivalent to the Givens Rotations solution. If the given matrix, *A*, is well-conditioned, the resulting matrices satisfy their required matrix characteristics. The algorithm for QRD-MGS is shown in Figure 2.

```
for $i = 1:n$
    $t_i = A_i$
for $i = 1:n$
    $R_{ii} = \|t_i\|$
    $Q_i = t_i / R_{ii}$
    for $j = i+1:n$
        $R_{ij} = Q_i^* t_j$
        $t_j = t_j - R_{ij}Q_i$
```

Fig. 2. MGS algorithm

After decomposition of the given matrix, back-substitution is performed to calculate the weights. The back-substitution stage has fewer calculations compared to QR decomposition; this algorithm is shown in Figure 3.

```
$x_n = \frac{c_n}{R_{nn}}$
for $i = n-1:1$
    $x_i = \frac{1}{R_{ii}}\left[c_i - \sum_{j=i+1}^{n} R_{ij}x_j\right]$
```

Fig. 3. Back-substitution algorithm

IV. Fixed point arithmetic and Error Analysis

Fixed point arithmetic is important as it results in faster smaller, functional units. However, it can results in less accurate results if it is not carefully designed. In this section, we discuss these tradeoffs and formulate an appropriate fixed point representation that provides results that are similar to a floating point implementation, yet they are much faster and smaller.

A. *Fixed Point Arithmetic*

There are two different types of approximations for real numbers: fixed-point and floating-point numeration systems [13]. The floating-point arithmetic allows us to represent very large range of numbers with some constant relative precision. Fixed-point arithmetic represents a reduced range of numbers with a constant absolute precision. Usage of floating point arithmetic is expensive in terms for 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. Our design uses two's complement fixed point arithmetic, which is shown in Figure 4. The data lines used in implementation for fixed point arithmetic consist of an integer part, a fractional part and a sign bit.

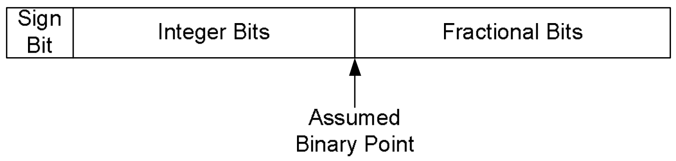


Fig. 4. Two's complement fixed-point representation is used in the calculations of the AWC core. The limited nature of the representation leads round-off and truncation errors. These errors must be investigated carefully for the precision analysis.

QR decomposition requires the usage of addition, subtraction, multiplication, division and square root operations. Number of calculations increases as the matrix dimensions grows. This is shown for different matrix sizes in Figure 5.

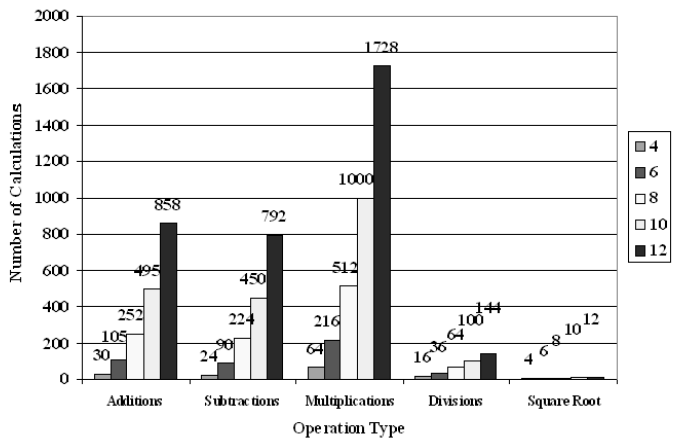


Fig. 5. Number of calculations for different matrix sizes, 4 x 4, 6 x 6, 8 x 8, 10 x 10 and 12 x 12 in terms of different operations. Number of calculations increases for larger matrices which affects the precision.

Figure 5 shows that decomposing a matrix into orthogonal and upper triangular matrices requires high number of operations. Some of these calculations are straightforward such as addition, subtraction and multiplication; however, division and square root operations are complex and can affect the precision significantly. Furthermore, they are inefficient in terms of FPGA implementation. Fixed-point arithmetic reduces precision 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 length after a computation. Truncation error occurs due to the limited number of bits to represent numbers. These issues must be handled carefully to prevent overflow which leads to incorrect results. Error analysis is a crucial step and will be considered in the next section.

B. *QR Decomposition Error Analysis*

While performing arithmetic calculations, a result may not fit into the reserved bits and if this case is not handled carefully, it causes overflow and incorrect results. To prevent overflow, every entry in the given matrix is normalized as the first step before starting decomposition. Normalizing is performed by dividing every entry by the largest matrix entry. Therefore, the given matrix entries are always in the [0, 1]. Normalization allows us to calculate the maximum number of bits required for the biggest possible integer. We then reserve this number of bits to insure that overflow does not occur. Error analysis for precision is very important in hardware designs especially when the arithmetic units use finite word length. We chose to evaluate QRD results for the error analysis due to its complexity compared to the back-substitution process. Also, these analysis results are useful in the evaluation of other designs which use QRD. The error analysis is performed by comparing the hardware simulation results with the actual results. The hardware simulation results are derived using Modelsim software by simulating our proposed architecture while the actual results are derived using Matlab software by simulating the decomposition using 16 bits floating point arithmetic. The error analysis is done for three different matrix sizes: 4 x 4, 6 x 6 and 8 x 8. Three different metrics are used for error analysis: mean error, standard deviation of error, and the mean percentage error.

The first metric, mean error, is computed by finding the error for all entries and then dividing the sum of these errors by the total number of entries. This calculation can be seen as

$$\sum_{i=1}^{m}\left|y_i-\hat{y}_i\right|/m \qquad (4)$$

where y, $\hat{y}$ and m are the actual results, the computed results and the number of entries which are used in the decomposition, respectively. Mean error is an important metric for error analysis however it does not include the information about outliers. 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

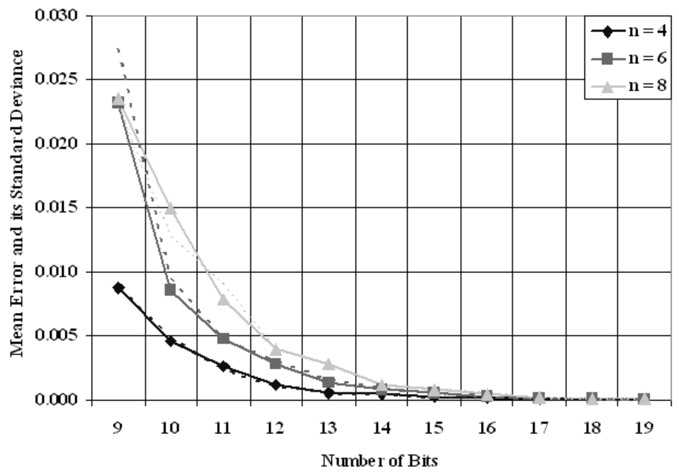


Fig. 6. The mean error and its standard deviation are important metrics for error analysis. For better precision, more bits must be used in calculations which introduce the tradeoff between the precision and area of the design. The solid lines and dotted lines represent mean error and standard deviation of the error respectively. The error and its standard deviation become significant if the number of bits used is less than 14.

the mean error, a second metric, the standard deviation of error, is used. The results of the two metrics are shown in Figure 6. The error depends on the number of bits used as data length for representing the numbers. There is a negative relationship between the error and the data length and this introduces a trade-off between precision and area of the design. It is important to notice that there is a cut-off point where the data length is 14. Precision doesn't improve significantly by using more bits than 14. More area efficient designs can be implemented using smaller data lengths where more accurate design can be implemented using bigger data lengths. Besides, the second metric shows that there is no significant effect of outlier errors where the data length is 14 or bigger.

Mean error sometimes leads to misleading conclusions because of the small range of numbers after normalization process. Therefore the third metric, mean percentage error, makes more sense if the relative error is considered. This metric is defined as

$$\sum_{i=1}^{m}\left|\left(\frac{y_i - \hat{y}_i}{y_i}\right)\right| / m \tag{5}$$

It considers the error compared to the actual result. The mean percentage error is increasing significantly for the number of bits less than 14 which is the same observation as before. These results are shown in Figure 7.

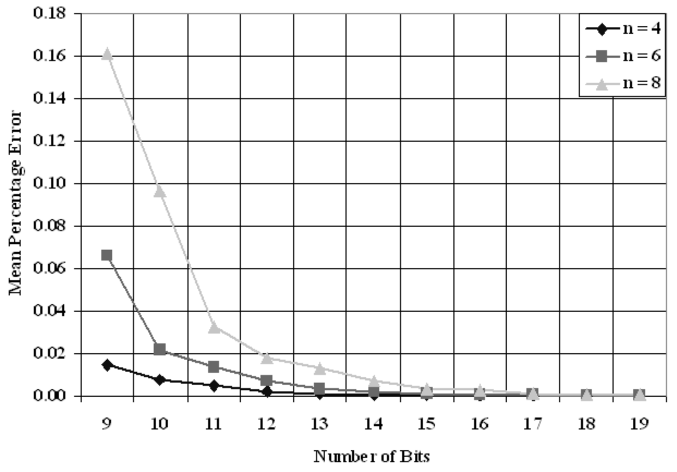


Fig 7. The mean error for error analysis can be misleading due to usage of normalized numbers which are always in the [0, 1]. The mean percentage error leads to more accurate conclusions if the relative error is considered.

V. Architectural Design of AWC Core

The proposed architecture works at the instruction-level where the instructions define the required calculations for the computation of weights of the system. 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. Dynamic Scheduling using Tomasulo's approach is a widely known approach to produce instruction level parallelism in presence of dependences [14]. In this approach, controller units track the operands to determine whether they are available and perform register renaming which assigns a free arithmetic unit for the desired calculation. Register renaming is provided by reservation station usage in every arithmetic unit where reservation stations fetch and buffer an operand as soon as the operand is ready. Our proposed design consists of two controller units and three arithmetic units. The arithmetic units are capable of computing decomposition and back-substitution. The control units are instruction & timing and operand controller. The arithmetic units are adder/subtractor, multiplier/divider and square root units. The proposed AWC core is shown in Figure 8.

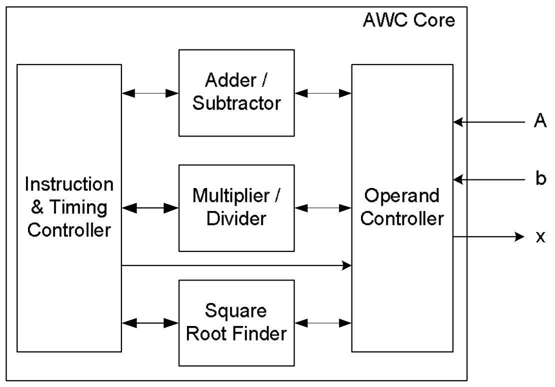


Fig. 8. The proposed AWC core is capable of doing QRD and back-substitution. AWC core consists of 2 controller and 3 arithmetic units.

The Instruction & Timing controller unit keeps the instructions which are required to perform AWC. The control elements supervise the status of the reservation stations of the arithmetic units and the current place of the operands. The Instruction & Timing controller function is as follows: 1) it gets the instructions from the memory element by ensuring the maintenance of correct data flow, 2) decodes the instruction to understand the desired calculation, the required operands and the destination memory entry in operand controller unit, 3) creates scheduled instructions by performing register renaming, 4) checks the current status of the arithmetic units and the current place of the operands and sends the scheduled instruction to the other units of the design or stall the AWC if there is no free reservation station.

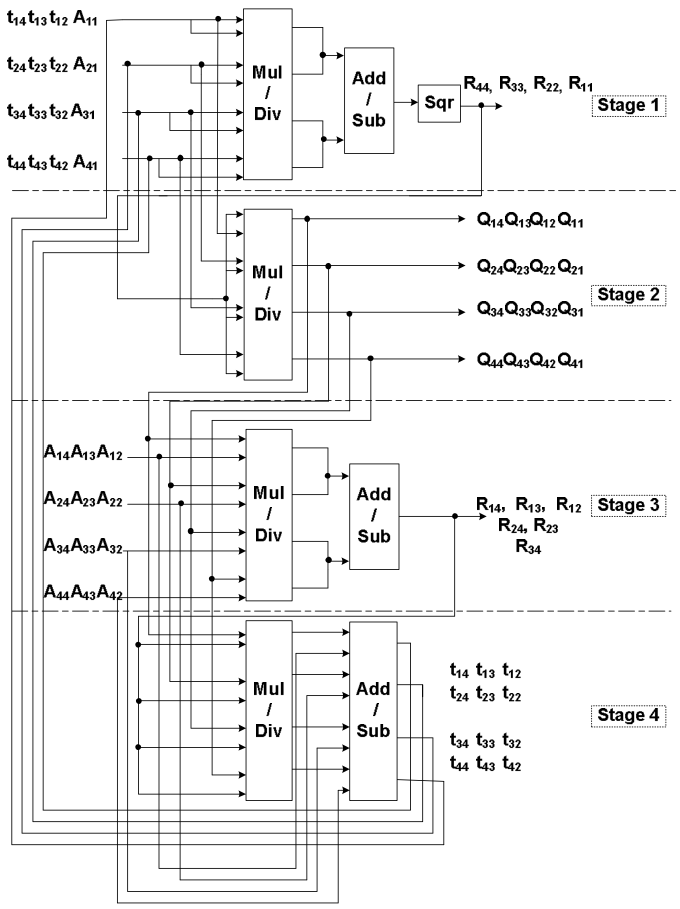
Every AWC starts by storing the given matrix data, *A*, and vector *b* into the memory entries of the Operand Controller. The Operand Controller sends the required operands for calculations if the operand is ready. Otherwise it waits until the calculation of the operand is completed and then sends the data after updating the memory.

Arithmetic units are reserved for specific calculations. The arithmetic units are adder/subtractor, multiplier/divider and square root finder. Each unit, except the square root unit, consists of matrix size number of reservation stations to perform desired calculations concurrently. Every arithmetic unit has three different stages: fetching the instruction from Instruction & Timing Controller, fetching the required operands from the outputs of the arithmetic units or from the memory unit of Operand Controller unit and performing the calculation. Addition, subtraction and multiplication use one clock cycle. Division and square root operations use *N* clock cycles, where *N* is equal to the number of bit in the input data.

If the input matrix is 4 x 4, AWC core uses 4 reservation stations in each adder/subtractor and multiplier/divider units due to the fact that they are able to perform these calculations in parallel. The Square root function unit uses only one reservation station because it is rarely needed during the QR decomposition. Figure 9 depicts our proposed architecture for scheduled arithmetic units to calculate QRD in AWC core. There are 4 different stages for computing QRD and every stage follows its next one like a thread and the stage resources are the same because of the usage of scheduling. The solution of back-substitution uses the same architecture with different scheduling.

VI. FPGA Implementation results

The proposed architecture is implemented on a Virtex4-xc4vsx35 FPGA. We use 14 bits for the data representation and also compare its results with the usage of 19 bits. The addition and subtraction functional units are implemented by using SLICEs with low delay carry chain network, the multiplications use XtremeDSP blocks which are very efficient for this purpose, divider core creates a circuit for fixed-point division based on radix-2 non-restoring division and finally square root function uses CORDIC core, all provided by Xilinx Coregen toolset.


$t_{14}t_{13}t_{12}\,A_{11}$
$t_{24}t_{23}t_{22}\,A_{21}$
$t_{34}t_{33}t_{32}\,A_{31}$
$t_{44}t_{43}t_{42}\,A_{41}$
Mul / Div
Add / Sub
Sqr
R_{44}, R_{33}, R_{22}, R_{11}
Stage 1
Mul / Div
$Q_{14}Q_{13}Q_{12}Q_{11}$
$Q_{24}Q_{23}Q_{22}Q_{21}$
$Q_{34}Q_{33}Q_{32}Q_{31}$
$Q_{44}Q_{43}Q_{42}Q_{41}$
Stage 2
$A_{14}A_{13}A_{12}$
$A_{24}A_{23}A_{22}$
$A_{34}A_{33}A_{32}$
$A_{44}A_{43}A_{42}$
Mul / Div
Add / Sub
R_{14}, R_{13}, R_{12}
R_{24}, R_{23}
R_{34}
Stage 3
Mul / Div
Add / Sub
t_{14} t_{13} t_{12}
t_{24} t_{23} t_{22}
t_{34} t_{33} t_{32}
t_{44} t_{43} t_{42}
Stage 4
Fig. 9. The proposed architecture for scheduled arithmetic units to calculate QRD is shown. The input is given *A* matrix and the results are *Q*, *R* or the updates of *A* matrices.

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 x 18 bit dedicated multipliers and 48-bit accumulator provide flexible resources to implement multipliers to achieve high performance. Furthermore, Xilinx Coregen software implements these cores very efficiently since it uses special mapping and place and route algorithms to implement the abovementioned cores to attain high performance design.

AWC core which uses 14 bits achieves 136 MHz of speed on the state of art Virtex4-xc4vsx35 FPGA with a throughput of 0.20M updates per second. The latency for AWC is 670 cycles. The resources used for AWC core are shown in Table 1. The design is easily extendable for larger matrix sizes by changing the instructions for scheduling which are required for AWC and using more block RAMs. For best timing results *n* number of reservation stations in arithmetic units are added in AWC core which is 4 in our design; and works for 4 x 4 matrices. It is important to state that more area efficient designs can be implemented using less reservation stations with the cost of more clock cycles to calculate the weights.

Table 1 – Resources for AWC core on a Virtex-4 FPGA using 14 bits for data representation

Design Unit	Area				
	LUTs	FFs	BRAMs	DSP48	SLICEs
Instruction & Timing Controller	179	45	1	-	93
Operand Controller	1310	360	-	-	747
Arithmetic Units	1780	1066	-	4	1052
Total	3,269	1,471	1	4	1,892

As we present in Section IV- B, using more bits for data representation leads to more accurate results however introduces more area. Figure 10 shows the comparison between using 14 and 19 bits in terms of area. AWC core which uses 19 bits achieves 130 MHz of speed on the same device with a throughput of 0.13M updates per second. AWC core which uses 19 bits also achieves better results than the previous works in terms of area and throughput. The increase in the area is not significantly high due to usage of scheduling for utilizing the resources.

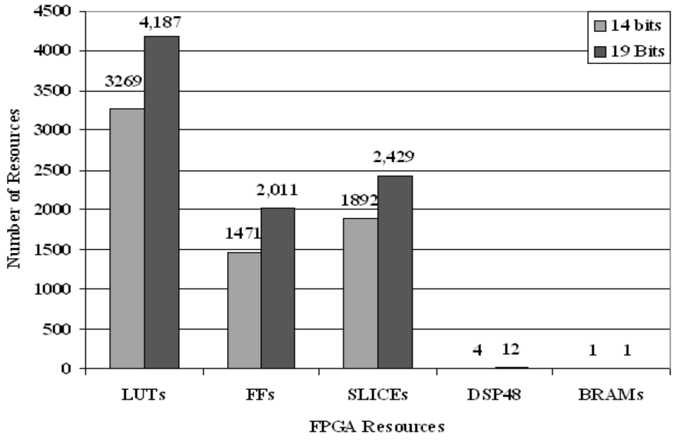


Fig. 10. The area comparison between using 14 and 19 bits for data representation. More accurate designs need more number of bits and leads to larger area results.

Table 2 shows the comparison between our design, [11] and [13] in terms of area and throughput. Karkooti et. al. presented a matrix inversion core using QRD-RLS algorithm [11] and we choose their results to compare with ours because of the similarity between the matrix inversion and AWC. The usage of squared Givens rotations and a folded systolic array is considered in their work. Number of resources is presented as slices, DSP48 and BRAMs, however FFs and LUTs are not given as total. Dick et. al is considered the design of a real-time QRD-based beamformer using GR[13]. Their design employs CORDIC-based processing and multiply-accumulate (MAC) based arithmetic. The throughput result is presented for 5 x 5 matrices. Both of these designs are implemented on a Virtex4 FPGA. Table 2 shows that our design results are better than the previous works. *NR* stands for the data which is not reported.

Table 2 - Comparison of Our Results with the Previous Work in terms of area and throughput. The term "NR" means that the results were not reported in the other publications.

	Area					Through put
	Slices	FF	LUT	DSP48	BRAM	Updates/ s
[11]	9117	*NR*	*NR*	22	9	0.13M
[13]	3530	5916	5411	13	6	0.09M
Our	1,892	1,471	3,269	4	1	0.20M

VII. Conclusion

An adaptive weight calculation (AWC) core is designed and implemented on Xilinx4-SX FPGA using QRD-RLS and Modified Gram-Schmidt algorithms with fixed-point arithmetic. The error analysis for QRD is investigated for different size of matrices and 14 bits of data length is found as a cut-off point between precision and area. The proposed design runs with a clock rate of 136 MHz and achieves a throughput of 0.20M updates per second. The proposed design has better results than the previous works in terms of area and throughput. The design is easily extendable to other matrix sizes.

References

[1] Simon Haykin, *Adaptive Filter Theory*, Prentice Hall, Fourth Edition.
[2] Jun Ma; Parhi, K.K.; Deprettere, E.F., *Annihilation-reordering look-ahead pipelined CORDIC-based RLS adaptive filters and their application to adaptive beamforming*, IEEE Transactions on Signal Processing, 2000.
[3] G.H. Golub and C. F. Van Loan, *Matrix Computations*, 3 rd ed. John Hopkins University Press, Baltimore, MD, 1996.
[4] Ray Andraka, *A Survey of CORDIC Algorithms for FPGA based computers*, International Symposium on Field Programmable Gate Arrays, 1998.
[5] Å, Björck, "Numerics of Gram-Schmidt Orthogonalization", Linear Algebra and Its Applications, 198:297-316, 1994.
[6] Singh, C.K.; Sushma Honnavara Prasad; Balsara, P.T., *VLSI Architecture for Matrix Inversion using Modified Gram-Schmidt based QR Decomposition*, 20th International Conference on VLSI Design, 2007.
[7] Lightbody, G.; Woods, R.; Walke, R., *Design of a parameterizable silicon intellectual property core for QR-based RLS filtering*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2003.
[8] Zhaohui Liu; McCanny, J.V.; Lightbody, G.; Walke, R., *Generic SoC QR array processor for adaptive beamforming*, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 2003.
[9] Lightbody, G.; Walke, R.; Woods, R.; McCanny, J., *Parameterisable QR core*, Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers, 1999.
[10] Lightbody, G.; Woods, R.; McCanny, J.; Walke, R.; Hu, Y.; Trainor, D., Rapid design of a single chip adaptive beamformer, IEEE Workshop on Signal Processing Systems, 1998.
[11] Karkooti, M.; Cavallaro, J.R.; Dick, C., *FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm*, Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005.
[12] Boppana, D.; Dhanoa, K.; Kempa, J., *FPGA based embedded processing architecture for the QRD-RLS algorithm*, 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004. FCCM 2004.
[13] Dick, Chris; Harris, Fred; Pajic, Miroslav; Vuletic, Dragan; *Real-Time QRD-Based Beamforming on an FPGA Platform*, Fortieth Asilomar Conference on Signals, Systems and Computers, 2006. ACSSC '06.
[14] Deschamps, J. P., Bioul G. J. A., Sutter G., *Synthesis of Arithmetic Circuits, FPGA, ASIC and Embedded Sytems*. John Wiley & Sons, Inc., 2006.
[15] John L.Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman Publishers, Third Edition.