Hardware and Software Design for QR Decomposition Recursive Least Square Algorithm

Sufeng Niu[1], Sizhou Wang[1], Semih Aslan[2] and Jafar Saniie[1]

[1]*Department of Electrical and Computer Engineering*
Illinois Institute of Technology
Chicago, Illinois

[2]*STARS Lab, Ingram School of Engineering*
Texas State University
San Marcos, Texas

Abstract — **In this paper, an embedded hardware and software system design and implementation for QR Decomposition Recursive Least Square (QRD-RLS) algorithm using Givens Rotation are presented. Furthermore, hardware and software design optimization are introduced to the Givens Rotation-based method. The computation performance is compared for hardware implementation running on Xilinx Virtex-5 FPGA, and software design running on two different processors (Intel i7 processor and ARM embedded processor) for solving least square problems. The challenges for hardware optimization and software algorithm are also presented.**

Keywords: Least square problem, QR decomposition, RLS, systolic, hardware and software.

I. Introduction And Background

Least square problems play a key role in adaptive signal processing applications such as wireless communication, channel equalization and bio-signal extraction [1]. Linear adaptive signal processing generally can be realized by two methods: Least Mean Square (LMS) and Recursive Least Square (RLS). Although both methods are able to process signals adaptively, the RLS algorithm has faster convergence and better performance than LMS algorithm [1] [3] [4], while LMS can be implemented by using less hardware.

The RLS algorithm is much more complex. When compared with the LMS algorithm which needs O(N) Multiplier and Accumulate (MAC) operations per sampling interval, RLS algorithm requires O(N^2) MAC operations per sampling interval, where N is the number of adaptive filter taps. Some expensive hardware such as division and square root are also required for RLS algorithm. To solve the RLS algorithm we have applied QR decomposition into the RLS algorithm [7] [8].

The QR decomposition can be accomplished by Gram–Schmidt Process, Householder Transformations, or Givens Rotation [2]. The Givens Rotation method has the most robust numeric property, and it is able to be mapped on systolic architecture proposed by Gentleman and Kung [5] to accelerate the computation speed.

In this paper, our goal is to realize a QRD-RLS algorithm into an embedded system using highly efficient hardware architecture and software design methodology. Hardware optimization is discussed for real-time signal processing purposes. Software design running on the Zynq platform [10] is also introduced for embedded application purposes. The organization of the paper is as follows: Section II provides the RLS algorithm backgrounds. Section III proposes the hardware design based on our previous work and software development for embedded systems. Section IV details the performance comparison between hardware and software approaches.

II. QRD RLS Algorithm And Overview

A. QRD RLS Overview

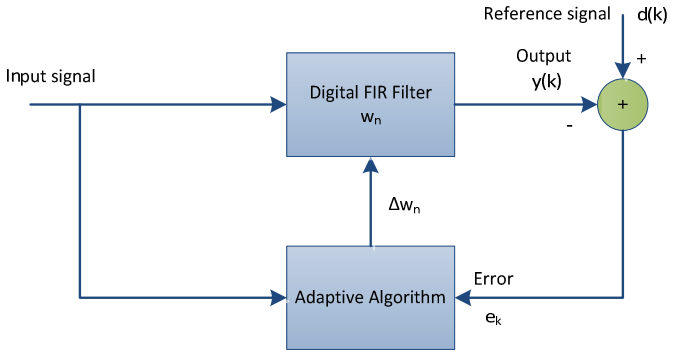

Figure 1. Adaptive filtering structure

The basic adaptive filtering concept is shown in Figure 1. We use the FIR model because it is more common than IIR in adaptive signal processing, since changing IIR coefficients adaptively may cause the filter to be unstable [1]. In Figure 1, the goal is to minimize the sum of square error as given by (1) and (2):

$$\min\left(\sum_{n=0}^{k} \lambda^{k-n} e_k^2\right) \quad (1)$$

where:

$$e_k = d_k - X_k w_n \quad (2)$$

978-1-4799-0066-4/13/$31.00 ©2013 IEEE

In the above equation, w_n is FIR coefficients, X_k is input matrix, d_k stands for reference-signal input vector and λ is called forgetting factor that determines the sensitivity of the filter to recent samples or previous samples. According to Ref. [6], we can derive the solution for least square problems as in (3):

$$w_n = [X_k^T X_k]^{-1} X_k^T d_k = X_k^{-1} d_k \quad (3)$$

For QR decomposition applied to a least square problem, we write

$$X_k = \mathrm{QR} \quad (4)$$

where Q is orthogonal matrix and R is upper triangular matrix. To solve (3), the QRD-based method is derived as:

$$Rw_n = Q^T d_k \quad (5)$$

By back substitution for R matrix, the weight vector can be extracted iteratively. This method avoids the matrix inversion operation to solve linear equations. The extracted weight vector, which is the FIR filter coefficients, is changed adaptively in every iteration k.

III. QRD RLS Implementation

A. Hardware Implementation based on Systolic Array

Hardware implementation provides highly desirable high speed circuit acceleration. Current reconfigurable logic devices make the hardware flexible and efficient. The purpose of hardware implementation is to realize high throughput and low latency signal processing algorithms. Thus, in this study, we focus on the circuit operational speed.

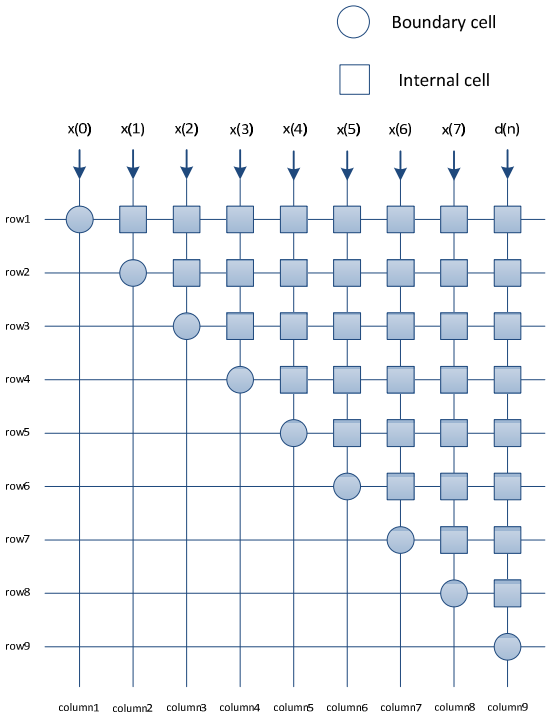

Figure 2. Systolic architecture for 8 by 8 matrix QR decomposition

The Givens Rotation algorithm is mapped onto a systolic array. In Figure 2, the boundary cell (circle) and internal cell (rectangle) of the triangular array are made to work fully in parallel. This parallel architecture is easily implemented with pipelining and suitable for ASIC fabrication. Nevertheless, the boundary cell contains more complex arithmetic computations: square root and division operation, while the internal cell consists of MACs. In our previous work the QR decomposition boundary cells were optimized by retiming and using look-up tables for fast arithmetic operations [7] [8]. The improved boundary cell design [8] is used for efficient arithmetic which offers a tradeoff between throughput and latency and is derived based on Newton's method.

In practice, to acquire a better filter performance, the application requires a higher number of filter taps. This means increasing the matrix dimensions. Assuming matrix dimension to be n, it requires $n(n+1)/2$ processing elements (PEs) for hardware implementation. Thus, matrix decomposition based on systolic array becomes very expensive. However, the proposed folded version of [9] systolic array as shown in Figure 3, reduces hardware resource consumption significantly.

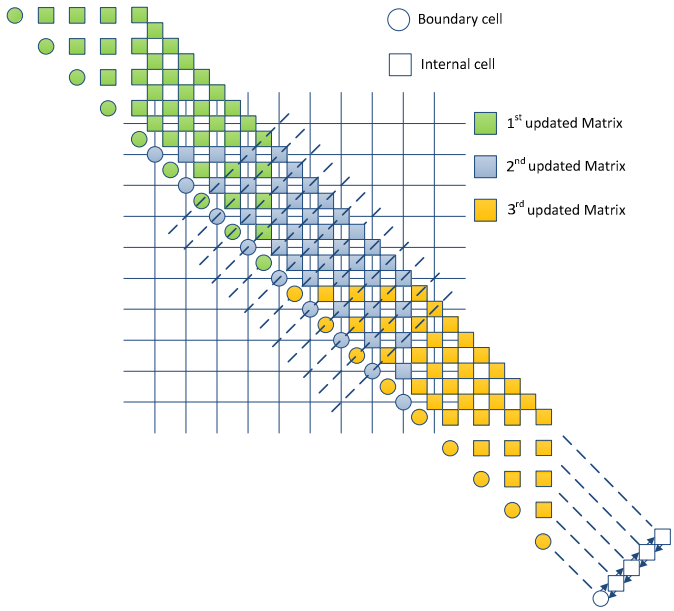

Figure 3. Linear mapping

Figure 3 shows the linear mapping of 8 by 8 matrix decomposition. The streaming data enable almost 100% utilization of folded PE resources, and the number of PEs is reduced from $n(n+1)/2$ to $(\lfloor n/2 \rfloor + 1)$. When the matrix dimension is even, the folded array fits completely. When the matrix dimension is odd, the bottom boundary cell in each matrix can be dropped to avoid computation overlapping, and dropping this operation will not affect weight computation results since the bottom boundary cell only affects output error e_k. To get accurate e_k, one more multiplier is required, which will not consume much hardware resources. Although the latency increases significantly, the circuit still keeps high throughput by similar optimization done in our previous work

[8]. Additional state machine is required to control the timing and data paths. Based on folded architecture and optimized PEs, the hardware module can achieve high throughput with smaller circuit area.

B. Software Design for Embedded System

In many embedded systems, applying reconfigurable logic for calculation may not be feasible due to the cost and power consumption. The goal of software design is to investigate state-of-the-art embedded processor capability for adaptive signal processing. Here, we examine the QR decomposition algorithm using the Intel i7 processor and also the embedded ARM processor on Zynq platform. To solve the least square problem, we developed a C code to calculate the R matrix. The program flowchart is shown in Figure 4:

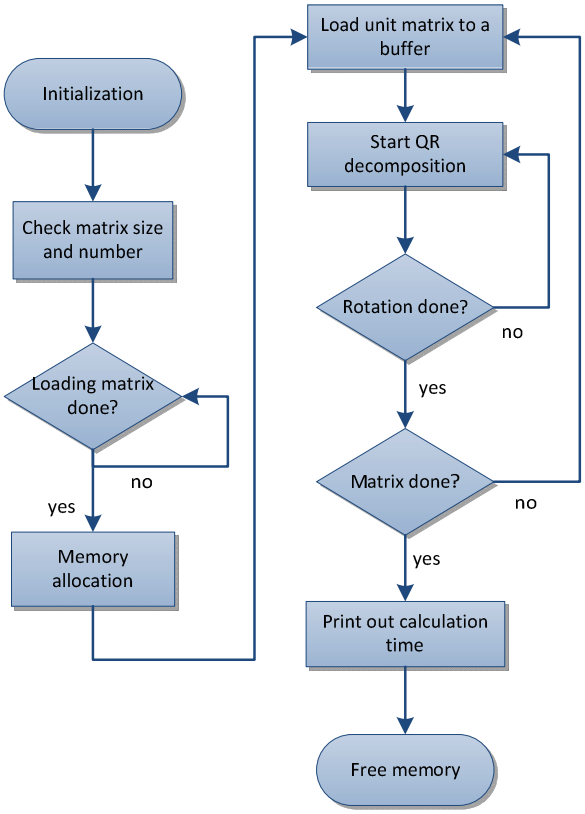

Figure 4. The QR decomposition software flowchart

In each loop, the rotation function performs:

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{6}$$

where:

$$\begin{aligned} c &= \cos\theta \\ s &= \sin\theta \end{aligned} \tag{7}$$

$\sin\theta$ and $\cos\theta$ are defined as:

$$\sin\theta = \frac{y_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \tag{8}$$

$$\cos\theta = \frac{x_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \tag{9}$$

The computation procedure for the Givens Rotation algorithm is presented in Equation (6) Through Equation (9) and the C-code is shown in Figure 5.

```
// i: column index, j: row index
for(i = 0; i < matrix_col; i++){
        for(j = matrix_row; j > i+1; j--){
                int k;

                a1 = data_in[i + (j-2)*matrix_col];
                a2 = data_in[i + (j-1)*matrix_col];
                // set breakpoint for debugging purpose, check a1, a2 value

                rotator = rotate(a1, a2); // rotate operation
                // set breakpoint, check rotator.c rotator.s value

                data_in_start = data_in + i + (j-2)*matrix_col;
                data_out_start = data_out + i +(j-2)*matrix_col;

                // copy to data_out mem space
                interval = matrix_col - i;
                matrix_trans(rotator, data_in_start, data_out_start, interval);
                // set breakpoint, verify data_in and data_out

                for(k = 0; k < interval; k++){
                        data_in_start[k] = data_out_start[k];
                        data_in_start[k+matrix_col] = data_out_start[k+matrix_col];
                }
        }
}
```

Figure 5. C code for Givens Rotation

The test vector is a group of random numbers generated from MATLAB. We import the test vector into the C code, and we use standard libraries to implement it. Therefore, we are able to use the same code with different compilers to get different bin files for different embedded processors.

Zynq integrates dual-core ARM Cortex-A9 and reconfigurable logics. The device is designed for System-on-Chip (SoC) applications [10]. The Neon core, available within Zynq, is a Single Instruction Multiple Data (SIMD) processor which helps to improve vector processing performance. In the makefile, we enable Neon processor for code optimization using cross compiler.

Zynq system was built with the Xilinx XPS tool and Linux Operating System (OS). We use the buildroot tool to build embedded Linux on the Zynq board. Buildroot is a set of makefile and patches to build embedded Linux in a very simple way. It can generate cross-compilation tool chain, Linux kernel, root file system and bootloader based on the desired configurations. The Secure Digital (SD) memory card is mounted to store the program binary file and test vector file.

IV. Performance Comparison And Simulation

We test QR decomposition for 4 by 4 matrix size. The input vectors are generated by random function Application Programming Interface from MATLAB tools. We synthesize the systolic array VHDL code with Xilinx XUPV5 platform. This RTL code is cycle accurate so that every matrix processing can be done in 56 clock cycle, running at 200MHz clock frequency.

The software acquires the calculation time using clock function. We have to repeat this multiple times to get an

accurate time measurement because a single loop is too short to measure. The computer system is Intel i7-950 running at 3.06GHz, and the Zynq platform is ARM Cortex-A9 dual-core working at 533MHz. Both platforms are running Linux OS. Each data is executed three times with different matrix number. The processing time measurements are shown in Figure 6 and Table I.

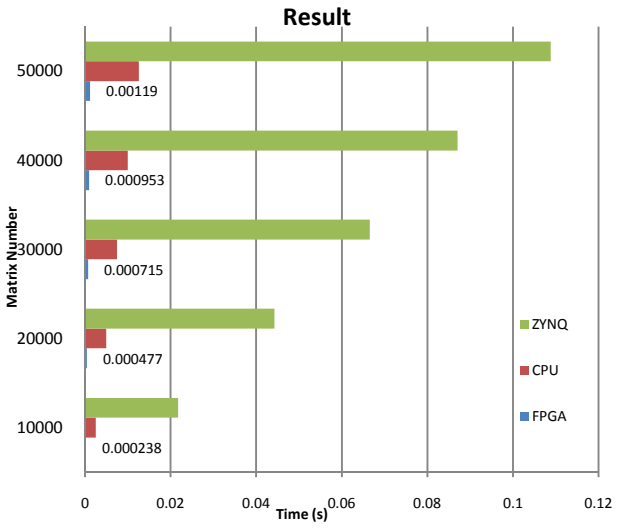

Figure 6. Performance comparison

TABLE I. PERFORMANCE COMPARISON

Platform	Number of Matrix Operations				
	10,000	*20,000*	*30,000*	*40,000*	*50,000*
FPGA	0.000238	0.000477	0.000716	0.000954	0.001192
Intel i7	0.002487	0.00498	0.00748	0.001001	0.001257
Zynq	0.02176	0.0442	0.06652	0.087067	0.108807

Unit: second

In Table I, measurements are taken for matrix number 10,000 to 50,000. It clearly illustrates that the FPGA runs nearly 10 times faster than the Intel i7 core, while the Intel i7 core is almost 10 times faster than ARM core.

The reason why FPGA runs so fast is that the VHDL code is optimized by deep pipelining. The streaming latency for each matrix updates is 5 clock cycles. When running at 200MHz, the latency of single matrix decomposition is approximately 0.27 seconds. For the Intel i7 processor, the average latency for one matrix decomposition is 0.24 seconds. CPU design is able to provide lower latency due to the high frequency of the CPU (3.06GHz) and good ASIC properties. Reconfigurable logic benefits from high hardware throughput so that even when running at 200MHz, it can provide the best computational performance. Another reason is that the FPGA is implemented as a 16-bit fixed point number representation, and CPU-based designs are implemented as a 64-bit floating point representation.

ARM core performance is actually good enough to provide adaptive signal processing in many embedded system applications. Additionally, we use the Xilinx Power Estimator tool to roughly estimate FPGA and Zynq power consumption. Viretx-5 FPGA static power consumption is 1.248 Watt, and Zynq consumes 1.026 Watt in total as shown in Figure 7, which means Zynq has the lowest cost and power consumption among these platforms.

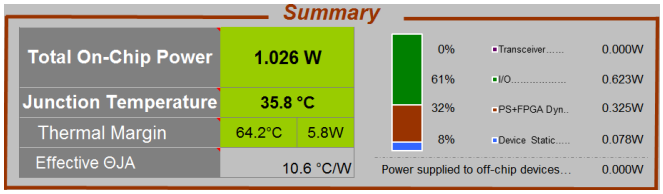

Figure 7. Xilinx Power Estimator for Zynq

V. CONCLUSION

In this paper, we estimate the matrix decomposition for the least square problem. The improved hardware design and C-based software design are discussed. The folded hardware architecture makes high dimension matrix decomposition possible. The software design provides power efficient and low cost approach for embedded system. The computation speed and capability are discussed as well.

REFERENCES

[1] B. Widrow; S.D. Stearns; , *Adaptive Signal Processing*, Prentice Hall, 1985

[2] G.H. Golub; C.F. Van Loan; , *Matrix Computations*, Baltimore, MD: Johns Hopkins, 1996

[3] D.T.L. Lee; M. Morf; B. Freidlander; , "RLS Ladder Estimation Algorithms," *IEEE Transaction on Acoustics, Speech and Signal Processing,* vol. ASSP-29, pp. 1362-1367, Jun. 1981, part III

[4] J.M. Cioffi; , "The Fast Adaptive ROTOR's RLS Algorithm," *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. 38, no. 4, pp. 631-653, Apr. 1990

[5] W.M. Gentleman; H.T. Kung; , "Matrix Triangularization by Systolic Arrays," in *Proceedings of SPIE- The International Society for Optical Engineering*, vol. 298, Real-Time Signal Processing IV, pp. 19-26, Aug. 25-28, 1981

[6] J.G. Proakis; D.G. Manolakis; , *Digital Signal Processing: Principles, Algorithm, and Applications,* Pearson, 2007

[7] S. Aslan; S. Niu; J. Saniie; , "FPGA Implementation of Fast QR Decomposition Based on Givens Rotation," *IEEE 55th International Midwest Symposium on Circuits and Systems,* pp.470-473, Aug. 5-8, 2012

[8] S. Niu; S. Aslan; J. Saniie; , "FPGA based High Speed Adaptive Signal Processing System," *in Proceedings of 2013 IEEE International Instrumentation and Measurement Technology*, pp. 1662-1665, May 6-9, 2013

[9] J.W. Luo; C.C. Jong; , "Scalable Linear Array Architectures for Matrix Inversion using Bi-z CORDIC," *Microelectronics Journal,* vol. 43, no. 2, pp. 141-153, Feb. 2012

[10] Xilinx. (2013) Zynq. [Online]. Available: http://www.xilinx.com

[11] T. Petazzoni, K. Kruse, N. Ludd, M. Herren. (2013) The Buildroot User Manual. [Online]. Available: http://buildroot.uclibc.org