Master's Thesis

Analysis and Implementation of Linear MIMO Signal Detection Algorithms

Dan Liu

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, February 2015.

Master's Thesis

Analysis and Implementation of Linear MIMO Signal Detection Algorithms

By

Dan Liu

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

Nowadays the demanding for wireless communication increases dramatically across the world. Multiple-input multiple-output (MIMO) is a wireless communication technology with great prospect. For baseband receivers, various sophisticated MIMO detection algorithms have been proposed in literature to exploit the gains provided by MIMO. However, it is challenging to implement algorithms on hardware platforms under different system operating scenarios.

In this thesis, two kinds of linear minimum mean square error (MMSE) MIMO detection algorithms are studied for MIMO systems with the same transmitting and receiving antenna numbers: squared MMSE and square-root MMSE. Specifically, three matrix inversion approaches are introduced to the former algorithm, including analytic or block-wise inversion, direct matrix inversion, and QR-decomposition (QRD) based inversion. As to the latter algorithm, QRD method is adopted.

First of all, computational complexity of the four schemes mentioned above are considered on two platforms which are application-specific integrated circuit (ASIC) and reconfigurable cell array (RCA) architecture. Comparisons show that QRD-based square-root MMSE requires the least operation amount on both platforms. Note that no matter which scheme is used, the operation amount on RCAs is higher than that on ASICs. However, the difference of complexity between two platforms decreases when antenna number increases.

Next, computation accuracy of the four aforementioned schemes is evaluated. Because of lower computational complexity, two QRD-based schemes are compared in 16-bit fixed-point format. Results show that square-root MMSE algorithm has a better performance.

Finally, due to the high computational accuracy and low complexity, QRD-based square-root MMSE is implemented using standard digital cell libraries and demonstrated on FPGA using 16-bit fixed-point format for a 4×4 MIMO system. Three different architectures are designed, including two pipeline versions and a time-multiplexed scheme. The throughput of the first pipeline version is 3 times faster than the second one, while its device utilization is twice as much as the latter. Time-multiplexed design has the lowest resource utilization; however, it reveals a lowest throughput because of the high data dependency in the QRD algorithm.

Acknowledgments

This Master's thesis would not exist without the support and guidance of my supervisor, Chenxin Zhang. I would like to express my sincere thanks to him. Also, I always appreciate his continuous encouragement and faith on me during this thesis work. In addition, I gratefully acknowledge my examiner, Joachim Rodrigues, for his instructions and advices on my thesis.

A special thank goes to my husband, Xin. He provides tireless help physically and mentally from the very beginning to the end of my study; he also gave me unlimited support and patience when I have difficulties moving forward. Besides, I need to say thank you to my son, Owen, for the infinite happiness he brings to me since he was born.

Finally, I would like to thank my friends, Zhonghua Wang, Ziyang Li and Hongwan Qin, for the stimulating discussions and great collaboration we had.

Dan Liu

Table of Contents

CHAPTER 1

1 Introduction

Multiple-input multiple-output (MIMO) is one of the main technologies in 3GPP Long Term Evolution Advanced (LTE-A). By exploiting spatial resources, MIMO improves channel capacity and link reliability without additional bandwidth.

However, comparing to single antenna systems, the computational complexity is significantly increased in MIMO systems, because of the sophisticated signal processing required to combat with inter-antenna interference. Therefore, implementation of MIMO signal processing has gained increasing attention in both academia and industry.

Application-specific integrated circuit (ASIC) is widely used in wireless communication system. However, it is designed to perform specific tasks [1]. By contrast, reconfigurable cell array (RCA) architecture is able to offer high flexibility by dynamic reconfigurations [2]. Hence, it is a promising platform to operate in diverse wireless communication scenarios.

This thesis focuses on two linear MIMO detection algorithms squared MMSE and square-root MMSE, and aims to implement one of them in a specific platform. Three matrix inversion approaches are introduced to solve the most computationally demanding part of the squared MMSE algorithm, that is, matrix inversion. For square-root MMSE, QR Decomposition (QRD) method is employed to simplify computation.

The contributions of the thesis are mainly in two aspects. First, computational complexity and accuracy of different MIMO detection schemes are analyzed. Computational complexity is evaluated based on ASIC and RCA platform respectively. For accuracy, both floating-point accuracy and fixed-point accuracy are taken into consideration. Second, on the basis of the comparison results, QRD-based square-root MMSE is chosen for hardware implementation due to its low complexity and high accuracy. Three fixed-point architectures are proposed with different performance in throughput and area consumption, including two pipeline versions and a time-multiplexed scheme.

1.1 MIMO system

Fig. 1.1 shows the block diagram of a typical MIMO system with N transmit and M receive antennas.

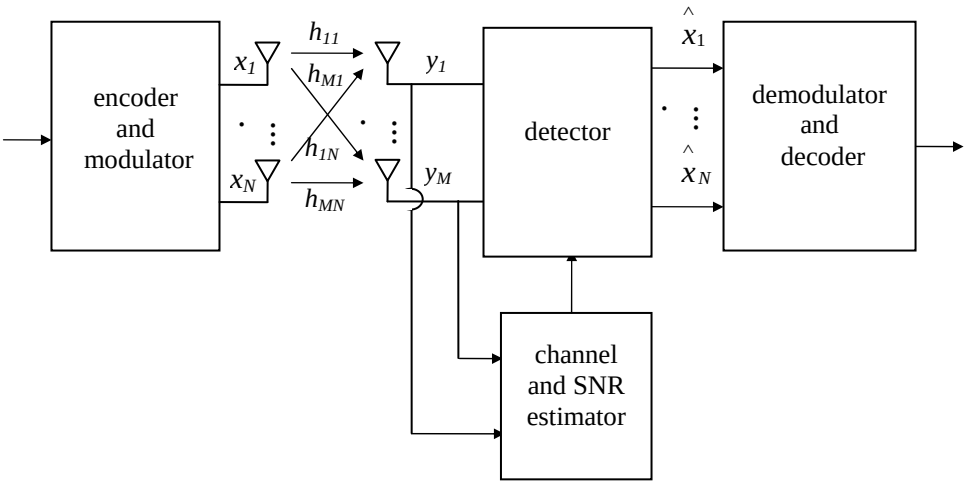

Fig. 1.1 MIMO system model with *N* transmit and *M* receive antennas

In such MIMO system, wireless channel model can be expressed by a *M*×*N* matrix *H* as (1.1) shows,

$$H = \begin{bmatrix} H_{1,1} & H_{1,2} & \dots & H_{1,N} \\ H_{2,1} & H_{2,2} & \dots & H_{2,N} \\ \dots & \dots & \dots & \dots \\ H_{M,1} & H_{M,2} & \dots & H_{M,N} \end{bmatrix}. \tag{1.1}$$

For each subcarrier, the $M \times 1$ received symbol vector *y* is given by (1.2)

$$y = Hx + n, \tag{1.2}$$

where *x* is the *N*×*1* transmitted symbol vector, and *n* is the *M*×*1* additive white Gaussian noise vector.

An *N*×*N* channel matrix means the MIMO system has the same number of transmit and receive antennas, for example, 2×2, 4×4 and 8×8 MIMO are commonly used configurations.

1.2 MIMO detection technology and algorithms

MIMO detector is used to estimate the transmitted symbol vector *x* from the received signal vector *y*, which is the main study subject of this thesis. There are various MIMO detection methods. Maximum likelihood (ML) detector provides the optimal performance, however its complexity dramatically increases when antenna number grows; by contract, linear detectors have low computational complexity but provide suboptimal performance [3], such as zero forcing (ZF) and minimum mean square error (MMSE) detectors. This thesis focuses on the linear MMSE detector which has the reasonable complexity and performance.

Varieties of algorithms exist to realize the linear MMSE MIMO detection strategy. Two linear MMSE MIMO detection algorithms are considered in the thesis, which are squared MMSE and square-root MMSE algorithm.

For squared MMSE, the estimation of transmitted signal is given by (1.3)

$$\hat{x} = (H^H H + \sigma^2 I_N)^{-1} H^H y = G_{MMSE} y, \tag{1.3}$$

where σ^2 is additive white Gaussian noise power, and I_N is the identity matrix of size *N*. For this function, matrix inversion is the main problem to solve. There are various approaches used to deal with matrix inversion, for example, QR decomposition (QRD) based inversion, analytic or block-wise inversion, and direct matrix inversion (DMI).

Since matrix inversion is a computational demanding operation, a square-root MMSE algorithm [4] was proposed which avoids matrix inversion by introducing a compound matrix. The disadvantage of this method is the increase of matrix dimension.

1.3 Reconfigurable architecture

Since ASICs have poor hardware reusability, reconfigurable architectures have been proposed, capable of implementing different algorithms onto the same platform. Fine-grained reconfigurable architectures, such as Field-programmable Gate Arrays (FPGAs), are able to provide “full” flexibility because of their bit-level manipulations. Compared with coarse-grained reconfigurable architectures, however, their performances are lower in terms of hardware utilization, power consumption, and configuration time [5]. As a result, although they have reduced flexibility due to word-level processing, coarse-grained reconfigurable architectures are increasingly considered in overhead-sensitive wireless communication systems.

This thesis is based on a kind of coarse-grained architecture, RCA, which consists of memory cells (MC) for data storage and processing cells (PC) to execute instruction, and their communication is via local connection and global network (GN) [2], as illustrated in Fig. 1.2.

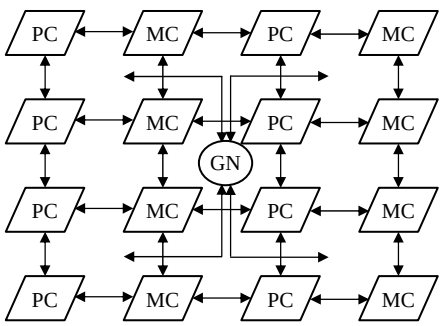

Fig. 1.2 An overview of a RCA

Specifically, Fig. 1.3 describes the structure of processing cell: several functional units (FU) are used to perform basic arithmetic operations, such as addition and multiplication of two complex-valued numbers, and by this way, several functional units can be used to process complex-valued matrix.

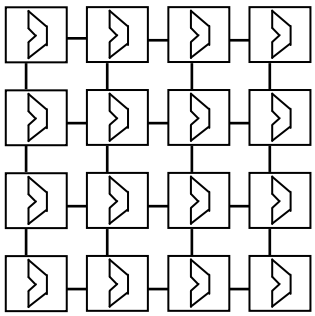

Fig. 1.3 PC composed of a FU array

1.4 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 analyzes computational complexity of different linear MMSE MIMO detection algorithms with the consideration of two different hardware platforms. Chapter 3 discusses the accuracy of algorithms in floating-point and fixed-point format respectively. How to realize QRD-based square-root MMSE algorithm on hardware platforms is given in Chapter 4. In detail, three fixed-point architectures are implemented, which are two pipeline versions and a time-multiplexed version. Comparisons among them are summarized in terms of throughput, device usage and processing speed. Finally, the conclusion of this research and suggestion of future work are presented in Chapter 5.

CHAPTER 2

2 Complexity Analysis of Linear MMSE MIMO Detection Algorithms

Matrices discussed in this thesis are composed of complex-valued numbers which require more computation resources than real numbers. Besides, matrix processing usually has high complexity. Therefore, it is necessary to compute the operation numbers of various algorithms in order to choose relatively simpler ones. In addition, hardware platforms also have some significant influence on algorithmic complexity. Thus, complexity analysis in this chapter will be based on two potential platforms: ASICs and RCAs.

For squared MMSE algorithm, the complexity analysis is mainly focused on different possible matrix inversion approaches involved in this algorithm. Three matrix inversion approaches are evaluated in terms of computational complexity. The operation amounts of square-root MMSE algorithm are exploited on the basis of a QRD method. Also, the comparison among the four linear MMSE MIMO detection schemes is presented on both platforms. This provides a reference for choosing algorithm and the corresponding hardware platform.

2.1 Complex numbers and complex arithmetic

Complex arithmetic has wide applications in the field of communications, signal processing, statistics, industry control, etc. Each complex number contains two real numbers. Hence, a complex-valued operation requires several steps of real-valued operations. Some examples are shown in (2.1)~(2.3). If $z_1 = a_1 + b_1 i$ and $z_2 = a_2 + b_2 i$, then

$$z_1 + z_2 = (a_1 + b_1 i) + (a_2 + b_2 i) = (a_1 + a_2) + (b_1 + b_2)i\,, \tag{2.1}$$

$$z_1 \times z_2 = (a_1 + b_1 i) \times (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + a_2 b_1)i\,, \tag{2.2}$$

$$z_1^* z_1 = (a_1 - b_1 i)(a_1 + b_1 i) = a_1^2 + b_1^2\,. \tag{2.3}$$

In the following discussion, the complexity of algorithms involving complex-valued arithmetic is evaluated based on the corresponding real-valued operations, since it is clear to compare the operation numbers of real-valued arithmetic.

2.2 Complexity of matrix inversion approaches in squared MMSE algorithm

As described in Chapter 1, for squared MMSE MIMO detection, the difficulty is to solve the inversion of matrix *V* which is defined in (2.4)

$$\hat{x} = (H^H H + \sigma^2 I_N)^{-1} H^H y = V^{-1} H^H y = G_{MMSE} y. \tag{2.4}$$

Therefore, in the following discussion, three matrix inversion approaches are evaluated in terms of complexity on ASICs and RCAs, including analytic or block-wise inversion, direct matrix inversion (DMI), and QRD-based inversion.

2.2.1 Complexity of a pre-process procedure

In addition to investigating matrix inversion approaches, it is necessary to calculate the operation numbers of computing matrix *V* before inverting it. This is because some matrix inversion methods require the computation of *V*, while DMI does not. To fairly compare with DMI, the other two inversion methods have to count this part of operations into their total complexity.

Define channel matrix *H* as (2.5)

$$H = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1N} \\ h_{21} & h_{22} & \dots & h_{2N} \\ \dots & \dots & & \dots \\ h_{N1} & h_{N2} & \dots & h_{NN} \end{bmatrix}, \tag{2.5}$$

where N is the antenna number (this thesis focuses on commonly used antenna setups, which are N=2, 3, 4, 8), and h_{ij} is a complex number. Then,

$$H^H H = \begin{bmatrix} h_{11}^* h_{11} + h_{21}^* h_{21} + \dots + h_{N1}^* h_{N1} & h_{11}^* h_{12} + \dots + h_{N1}^* h_{N2} & \dots & h_{11}^* h_{1N} + \dots + h_{N1}^* h_{NN} \\ h_{12}^* h_{11} + h_{22}^* h_{21} + \dots + h_{N2}^* h_{N1} & h_{12}^* h_{12} + \dots + h_{N2}^* h_{N2} & & h_{12}^* h_{1N} + \dots + h_{N2}^* h_{NN} \\ \dots & & & \dots \\ h_{1N}^* h_{11} + h_{2N}^* h_{21} + \dots + h_{NN}^* h_{N1} & h_{1N}^* h_{12} + \dots + h_{NN}^* h_{N2} & \dots & h_{1N}^* h_{1N} + \dots + h_{NN}^* h_{NN} \end{bmatrix}$$

$$\overset{define}{=} M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1N} \\ m_{21} & m_{22} & \dots & m_{2N} \\ \dots & \dots & & \dots \\ m_{N1} & m_{N2} & \dots & m_{NN} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{21}^* & \dots & m_{N1}^* \\ m_{21} & m_{22} & \dots & m_{N2}^* \\ \dots & \dots & & \dots \\ m_{N1} & m_{N2} & \dots & m_{NN} \end{bmatrix}, \tag{2.6}$$

where *M* is a Hermitian matrix.

Firstly, consider the complexity of calculating *V* on ASIC platforms. To attain every diagonal element m_{ii}, there are *N* multiplications of a complex number and its conjugate. This leads to a real-valued m_{ii}. By contrast, *N* complex-valued

multiplications are involved to compute non-diagonal element m_{ij}. Taking advantage of dedicated hardware design, ASIC approach is able to deal with computations of diagonal element and non-diagonal element in a different way. Therefore, the amount of real operations for m_{ii} is less than that of m_{ij}, derived from (2.1) ~ (2.3). Since M is a Hermitian matrix, m_{ji} can be obtained from a simple conjugate operation of m_{ij}. This property is used to reduce processing complexity by almost half. The total operation numbers for the calculation of V using ASIC platforms are listed in Table 2.1.

Table 2.1 Computational complexity of calculating V on ASIC and RCA platforms

			real multiplication	real addition	conjugate
ASICs	H^H				N^2
	$M = H^H H$	diagonal	$2N^2$	$2N^2 - N$	
		non-diagonal	$2N^3 - 2N^2$	$2N^3 - 3N^2 + N$	$\frac{N^2 - N}{2}$
	$V = M + \sigma^2 I$			N	
	total		$2N^3$	$2N^3 - N^2 + N$	$\frac{3N^2 - N}{2}$
RCAs	H^H				N^2
	$M = H^H H$		$2N^3 + 2N^2$	$2N^3 + N^2 - N$	$\frac{N^2 - N}{2}$
	$V = M + \sigma^2 I$			N	
	total		$2N^3 + 2N^2$	$2N^3 + N^2$	$\frac{3N^2 - N}{2}$

Secondly, map the calculation of matrix V on RCA platforms. In RCAs, every element of matrix M is treated in the same way, because FUs are designed for general purpose usage. The required operation for each element is the same as that m_{ij} in ASICs. Similar to ASICs, only $N(N+1)/2$ elements needs to be calculated, given the conjugate property of the Hermitian matrix. The computational complexity for calculating V on RCA platforms is shown in Table 2.1.

2.2.2 Complexity of analytic and block-wise matrix inversion approach

The analytic inversion method [6] for an $N \times N$ matrix V is described in (2.7)

$$V^{-1} = \frac{adj(V)}{|V|} = \frac{C^T}{|V|}, \tag{2.7}$$

where $|V|$ is the determinant of matrix V, and $adj(V)$ is the transpose of the matrix

of cofactors, known as the adjugate matrix. The element C_{ij} of adjugate matrix can be obtained by deleting the i^{th} row and j^{th} column of V and then taking the determinant of the $(N\text{-}1)\times(N\text{-}1)$ matrix times by $(-1)^{(i+j)}$.

Analytic inversion is an efficient method for low-dimension matrices, because it is easy to compute the determinant and the adjugate matrix for small matrices. However, for large matrices, this recursive analytic method requires a huge amount of calculation and an alternative method, block-wise inversion [7], is better to be adopted.

Block-wise inversion solves matrix inversion by inverting some small sub-matrices and then doing some multiplications and additions. The matrix inversion lemma is as follows

$$V^{-1}=\begin{bmatrix}A & B\\ C & D\end{bmatrix}^{-1}=\begin{bmatrix}A^{-1}+A^{-1}B(D-CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1}\\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1}\end{bmatrix}, \quad (2.8)$$

where A, B, C and D are sub-blocks of arbitrary size, and A and D must be square matrices [8]. Because it can reduce the amount of operations by partitioning a large matrix into smaller ones, block-wise inversion can be used to process large matrices [7].

1. Analytic inversion for small matrices

In this thesis, analytic inversion is used in 2×2 and 3×3 MIMO systems. If V is a 2×2 matrix, then

$$V=\begin{bmatrix}a & b\\ c & d\end{bmatrix} \quad \Rightarrow V^{-1}=\frac{1}{ad-bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}. \quad (2.9)$$

Because V is a Hermitian matrix, the denominator is a real number and the inversion of V is a Hermitian matrix too. The complexity counting is very simple and the results based on real numbers are listed in Table 2.2.

Table 2.2 Complexity comparison of 2×2 analytic matrix inversion on two platforms

	real multiplication	real addition	real division	conjugate
ASICs	7	3	1	1
RCAs	14	7	1	1

For the 3×3 matrix case, the inversion of V can be obtained using (2.10)

$$V^{-1}=\begin{bmatrix}a & b & c\\ d & e & f\\ g & h & i\end{bmatrix}^{-1}=\frac{1}{a(ei-fh)-b(id-fg)+c(dh-eg)}\begin{bmatrix}a & d & g\\ b & e & h\\ c & f & i\end{bmatrix}. \quad (2.10)$$

Consider the Hermitian property of matrix V,

$$V^{-1} = \frac{1}{aei - aff^* - ibb^* - ecc^* + 2\times \mathrm{Re}(bfc^*)} \begin{bmatrix} a & b^* & c^* \\ b & e & f^* \\ c & f & i \end{bmatrix}. \quad (2.11)$$

Table 2.3 compares the computational complexity of 3×3 matrix analytic inversion on the two platforms.

Table 2.3 Complexity comparison of 3×3 analytic matrix inversion on two platforms

	real multiplication	real addition	double of a real number	real division	conjugate
ASICs	46	30	1	1	3
RCAs	84	64	0	1	3

2. Block-wise inversion for large matrices

For MIMO system with more than 3 receiving/ transmitting antennas, block-wise inversion is used.

When N is 4, V can be divided into four 2×2 matrices,

$$V = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & v_{44} \end{bmatrix} = \begin{bmatrix} v_{11} & v_{21}^* & v_{31}^* & v_{41}^* \\ v_{21} & v_{22} & v_{32}^* & v_{42}^* \\ v_{31} & v_{32} & v_{33} & v_{43}^* \\ v_{41} & v_{42} & v_{43} & v_{44} \end{bmatrix} = \begin{bmatrix} A_{2\times 2} & B_{2\times 2} \\ C_{2\times 2} & D_{2\times 2} \end{bmatrix}. \quad (2.12)$$

According to (2.8), only two 2×2 matrix inversions need to be solved; both inversions can be easily solved by using the simple 2×2 analytic inversion method as shown in (2.9). In addition, the conjugate property of the Hermitian matrix should be considered for simplifying computations on the two platforms. The operation numbers when using RCA and ASIC platforms are listed in Table 2.4.

Table 2.4 Complexity comparison of 4×4 block-wise matrix inversion on two platforms

	real multiplication	real addition	real negated	real division	conjugate
ASICs	134	96	10	2	10
RCAs	172	132	10	2	10

For N=8, matrix V is divided into four 4×4 matrices

$$V = \begin{bmatrix} A_{4\times 4} & B_{4\times 4} \\ C_{4\times 4} & D_{4\times 4} \end{bmatrix}. \quad (2.13)$$

The inversion of A and $(D - CA^{-1}B)$ can be solved iteratively by using 2×2

block-wise inversion method, since they are 4×4 matrices. The operation numbers for RCA and ASIC platforms are summarized in Table 2.5.

Table 2.5 Complexity comparison of 8×8 block-wise matrix inversion on two platforms

	real multiplication	real addition	real negated	real division	conjugate
ASICs	1060	1080	52	4	60
RCAs	1432	1256	52	4	60

2.2.3 Complexity of QRD-based matrix inversion approach

QR decomposition is a special factorization method and plays an important role in linear equations, least squares and eigenvalue problems [9]. By QR decomposition, matrix A is factorized by the multiplication of an orthogonal matrix Q and an upper triangular matrix R, expressed as

$$A = Q \bullet R \,. \tag{2.14}$$

Hence, the inversion of matrix A can be solved by (2.15),

$$A^{-1} = R^{-1} \bullet Q^{H} \,. \tag{2.15}$$

In this section, the QRD-based squared MIMO detection formulation is derived after explaining a QRD algorithm. Then the complexity of this QRD-based inversion method is calculated.

1. Introduction of QRD

There are several algorithms to realize QR decomposition, and this thesis adopted Gram-Schmidt orthogonalization. Its main procedure is to sequentially orthogonalize the columns of A. However, the classic Gram-Schmidt (CGS) process may lead to numerical instability due to round off errors [10]. As a result, the modified Gram-Schmidt method was proposed as Algorithm 1 describes [10],

for i=1 to N
  $R_{ii} = \|A_i\|$
  $Q_i = A_i / R_{ii}$
  for j=i+1 to N
    $R_{ij} = \langle Q_i, A_j \rangle$
    $P_{ij} = R_{ij} Q_i$
    $A_j = A_j - P_{ij}$
  end for
end for

Algorithm 1: Modified Gram-Schmidt algorithm

where $<Q_i, A_j>$ is the inner-product of column vectors A_j and Q_i, $\|A_i\|$ is the norm

of A_i, and A is an $N \times N$ matrix; R_{ij} denotes the i^{th} row and j^{th} column element of matrix R.

This algorithm involves a nested loop: the outer loop is to obtain normalized vectors Q_1, Q_2... Q_i; and the inner loop is to project A_j orthogonally onto the line spanned by Q_i and then update A_j by subtracting the projection from A_j.

2. QRD-based matrix inversion for squared MMSE MIMO detection

For squared MMSE MIMO detection formula given by (2.4), define A as V. Based on the aforementioned QR decomposition scheme, the estimated value of transmitted signal can be solved by the following QR decomposition steps

$$A^{-1} = V^{-1} = (H^H H + \sigma^2 I_N)^{-1} = (QR)^{-1} = R^{-1} Q^H,$$

$$G_{MMSE} = (H^H H + \sigma^2 I_N)^{-1} H^H = A^{-1} H^H = R^{-1} Q^H H^H, \qquad (2.16)$$

$$\hat{x} = G_{MMSE} y = R^{-1} Q^H H^H y.$$

The inversion of the upper triangular matrix R can be computed by the algorithm shown in Algorithm 2 [11].

```
(1)   for j=1 to N
(2)     for i=N downto 1
(3)       if i>j
(4)           w_ij=0
(5)       elseif i=j
(6)           w_ij=1/R_ij
(7)       else
(8)           w_ij = -( sum_{m=1}^{j-1} w_im R_mj ) / R_jj
(9)       end if
(10)    end for
(11)  end for
```

Algorithm 2: Inversion of an upper triangular matrix R

3. Complexity of QRD-based matrix inversion

This QDR-based inversion method can be divided into three parts: QR decomposition, inversion of an upper triangular matrix R, and matrix multiplication between R's inversion and Q^H.

The main computational complexity of this scheme comes from QR decomposition. As shown in Algorithm 1, MGS is an iterative algorithm. The outer loop is repeated for N times. The iteration number of inner loop depends on the outer loop. Specifically,

(1) After Q_1 is calculated, A_j ($j=2,3...N$) are processed to get corresponding R_{1j}, orthogonal projection P_{1j} and then update themselves. The updated sequence is denoted as $A_j\_upd\_1^{st}$ throughout the following discussion.

(2) Similarly, when Q_2 is computed, $A_j\_upd\_1^{st}$ ($j=3,4...N$) need to be dealt with for obtaining R_{2j}, P_{2j} and to be updated. The updated sequence in this step is denoted as $A_j\_upd\_2^{nd}$.

(3) The same principle is applicable to Q_i ($i=3, 4…N-1$).

Thus, the inner loop need to be executed $N(N-1)/2$ times. To invert an $N\times N$ matrix involved in squared MMSE algorithm, the complexity of QR decomposition on RCAs and ASICs are summarized in Table 2.6.

Considering the other two parts of this inversion scheme. Because R is an upper triangular matrix, its inversion matrix W is an upper triangular matrix too. This characteristic simplifies the inversion of R, and it is also utilized when analyzing the complexity of matrix multiplication WQ^H. Besides, the inversion of A is a Hermitian matrix, therefore, its conjugate property can further simplify the computation process on the two platforms.

The total complexity of QRD-based matrix inversion scheme is summarized in Table 2.6.

Table 2.6 Computational complexity of QRD-based matrix inversion on two platforms

		real multiplication	real addition	square root	real reciprocal
ASICs	$A=QR$	$4N^3-6N+4$	$4N^3-N^2-8N+6$	N	N
	$W=R^{-1}$	$\frac{2N^3-2N}{3}$	$\frac{2N^3-6N^2+7N-3}{3}$		
	$A^{-1}=WQ^H$	$\frac{2N^3+3N^2+N}{3}$	$\frac{4N^3+3N^2-N}{6}$		
	total	$\frac{16N^3+3N^2-19N+12}{3}$	$\frac{32N^3-15N^2-35N+30}{6}$	N	N
RCAs	$A=QR$	$4N^3$	$4N^3-N^2-N$	N	N
	$W=R^{-1}$	$\frac{2N^3+3N^2-5N}{3}$	$\frac{2N^3+3N^2+4N-3}{3}$		
	$A^{-1}=WQ^H$	$\frac{2N^3+6N^2+4N}{3}$	$\frac{4N^3+9N^2+5N}{6}$		
	total	$\frac{16N^3+9N^2-N}{3}$	$\frac{32N^3-3N^2+7N-6}{6}$	N	N

2.2.4 Complexity of direct matrix inversion approach

DMI is developed from the updating procedure of the gain in Kalman filters [12]. Starting from (2.17)

$$P^{(0)}=\frac{1}{N\sigma^2}I\,, \tag{2.17}$$

the next N iterative operations are processed as

$$P^{(j)} = P^{(j-1)}(I - \frac{H_j^H H_j P^{(j-1)}}{1 + H_j P^{(j-1)} H_j^H}), \tag{2.18}$$

where j is the iteration number, and the result $P^{(N)}$ is the inversion of matrix V.

This procedure can be divided into eight steps with the operation numbers listed in Table 2.7 for RCA and ASIC platforms.

Table 2.7 Computational complexity of DMI for an $N \times N$ matrix on two platforms

		real multiplication	real addition	real division
ASICs	(1) $x = H_j \times P^{(j-1)}$	$4N^2 - 2N$	$4N^2 - 2N$	
	(2) $y = x \times H_j^H$	$2N$	$3N - 1$	
	(3) $z = 1 + y$		1	
	(4) $u = x^H$		N	
	(5) $v = u \times x = x^H x$	$2N^2$	N^2	
	(6) $q = 1/z$			1
	(7) $w = q \times v$	N^2		
	(8) $P^{(j)} = P^{(j)} - w$		$\frac{3N^2 - N}{2}$	
	subtotal (each iteration)	$7N^2$	$\frac{13N^2 + 3N}{2}$	1
	total (N iterations)	$7N^3$	$\frac{13N^3 + 3N^2}{2}$	N
RCAs	(1) $x = H_j \times P^{(j-1)}$	$4N^2$	$4N^2 - 2N$	
	(2) $y = x \times H_j^H$	$2N$	$3N - 1$	
	(3) $z = 1 + y$		1	
	(4) $u = x^H$		N	
	(5) $v = u \times x = x^H x$	$2N^2 + 2N$	$N^2 + N$	
	(6) $q = 1/z$			1
	(7) $w = q \times v$	$N^2 + N$		
	(8) $P^{(j)} = P^{(j)} - w$		$\frac{3N^2 + N}{2}$	
	subtotal (each iteration)	$7N^2 + 5N$	$\frac{13N^2 + 7N}{2}$	1
	total (N iterations)	$7N^3 + 5N^2$	$\frac{13N^3 + 7N^2}{2}$	N

2.2.5 Complexity evaluation on different matrix inversion approaches

Table 2.8 summarizes the computational complexity of three matrix inversion approaches when antenna number N is 2, 3, 4, and 8 on both RCA and ASIC platforms. Note that conjugate operation and real-valued negation are all counted as real-valued addition. Operation numbers for computing matrix V are counted for analytic/block-wise inversion and QRD-based inversion, since the DMI scheme is able to obtain it inversion without computing V (see (2.17) and (2.18) in section 2.2.4).

Table 2.8 Computational complexity comparison of different matrix inversion approaches

approach	platform	RCA				ASIC			
	antenna number	N=2	N=3	N=4	N=8	N=2	N=3	N=4	N=8
analytic/ block-wise	real multiplication	38	156	332	2584	23	101	262	2084
	real addition	33	142	318	2548	23	93	254	2252
	division (1/real)	1	1	2	4	1	1	2	4
DMI	real multiplication	76	234	528	3904	56	189	448	3584
	real addition	66	207	472	3552	50	171	408	3296
	division (1/real)	2	3	4	8	2	3	4	8
QRD-based	real multiplication	78	242	548	4072	54	192	464	3772
	real addition	67	217	503	3887	43	167	419	3587
	division (1/real)	2	3	4	8	2	3	4	8
	square root	2	3	4	8	2	3	4	8

From Table 2.8, two points can be observed.

Firstly, consider the complexity in terms of different inversion approaches. Fig. 2.1 shows the amount of additions and multiplications required in each inversion approach. Note that division numbers are ignored, since they are too small to be presented in the same figures.

For RCA platforms as Fig. 2.1(a) illustrates, the analytic or block-wise scheme is the least computational demanding way of calculating the inversion of V regardless of the number of antennas. Its operation numbers of multiplication, addition and division are all around half of the QRD-based inversion and about 65% of the DMI method. Although DMI and QRD-based inversion require almost the same amount of operations, the latter involves square root operation and is thus more complicated to compute. This conclusion also applies to ASIC platforms as seen from Fig 2.1(b).

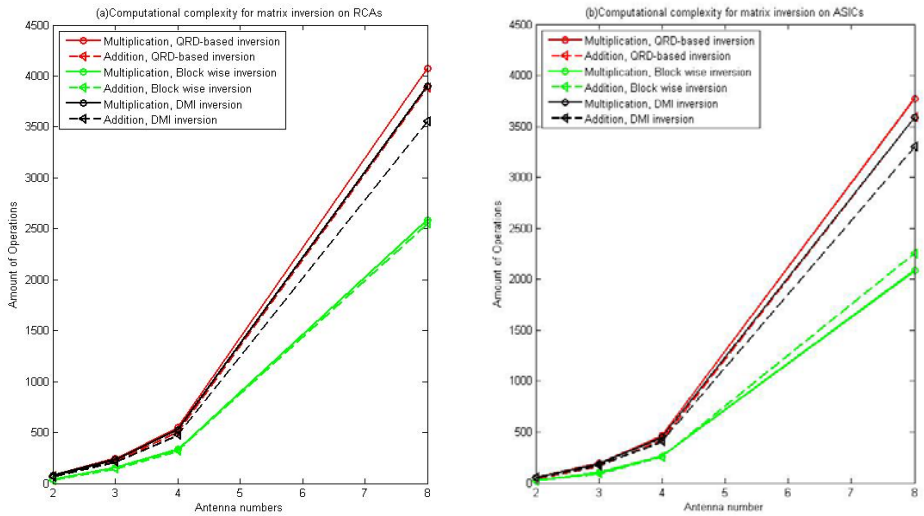

Fig. 2.1 Complexity comparison of matrix inversion approaches on RCAs and ASICs (multiplication: solid lines, addition: dashed lines)

Secondly, let's think about the impact of hardware platform on computational complexity. For a given antenna number, division operations are the same for every inversion approach regardless of the hardware platform. Besides, the number of square root operations required in the QRD-based inversion is the same for both platforms. The multiplication and addition numbers of each approach are less when using ASICs than those using RCAs. Although the operation amounts on both platforms increase when antenna number grows, the gap between different platforms decreases. For example, for each approach, multiplication and addition numbers on ASICs are between about 60% and 76% of those on RCAs when N = 2, while this ratio increases to up to 92% when N = 8.

2.3 Square-root MMSE MIMO detection based on QR Decomposition

For MMSE MIMO detection formula in (2.4), define transmitted signal $\underline{x}$ and received signal $\underline{y}$ by extending x and y with a null vector respectively, and an augmented channel matrix $\underline{H}$ by extending H with a noise covariance matrix, i.e.,

$$\underline{H} = \begin{bmatrix} H \\ \sigma I_N \end{bmatrix}_{2N \times N}, \qquad \underline{x} = \begin{bmatrix} x \\ 0_{N \times 1} \end{bmatrix}_{2N \times 1}, \qquad \underline{y} = \begin{bmatrix} y \\ 0_{N \times 1} \end{bmatrix}_{2N \times 1}. \tag{2.19}$$

Correspondingly, the MIMO detection formula can be rewritten as

$$A = H^H H + \sigma^2 I = \underline{H}^H \underline{H},$$

$$\hat{x}'_{MMSE} = (\underline{H}^H \underline{H})^{-1} \underline{H}^H \underline{y} = \underline{H}^+ \underline{y} = \hat{x}_{MMSE}, \tag{2.20}$$

where $\underline{H}^+$ is a pseudo inversion of matrix $\underline{H}$. Because pseudo inversion is computationally demanding, QR decomposition can be used to achieve a relatively simple method as (2.21)~(2.23) describe [5].

$$\underline{H} = \underline{Q}\,\underline{R} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \underline{R} = \begin{bmatrix} Q_1 \underline{R} \\ Q_2 \underline{R} \end{bmatrix} = \begin{bmatrix} H \\ \sigma I_N \end{bmatrix} \Rightarrow \begin{cases} Q_1 \underline{R} = H \\ Q_2 \underline{R} = \sigma I_N \Rightarrow \underline{R}^{-1} = Q_2 / \sigma \end{cases} \tag{2.21}$$

$$\Rightarrow \hat{x}'_{MMSE} = \underline{H}^+ \underline{y} = (\underline{Q}\,\underline{R})^+ \underline{y} = \underline{R}^{-1} \underline{Q}^+ \underline{y} = \frac{Q_2}{\sigma} \underline{Q}^+ \begin{bmatrix} y \\ 0_{N\times 1} \end{bmatrix}_{2N\times 1} = \frac{1}{\sigma} Q_2 Q_1^H y \tag{2.22}$$

$$\Rightarrow G'_{MMSE} = \frac{1}{\sigma} Q_2 Q_1^H \tag{2.23}$$

Here Q_1 and Q_2 are $N\times N$ matrices, and Q_2 is an upper triangular matrix with real-valued diagonal elements. It can be seen from (2.21) that the inversion of matrix $\underline{R}$ can be directly given by the result of QR decomposition. By doing this, computationally demanding matrix inversion is avoided. Hence, square-root MMSE scheme can be achieved by a QRD procedure for $2N\times N$ matrix $\underline{H}$ followed by multiplication of two $N\times N$ matrices and a real reciprocal.

Except different matrix size, this complexity analysis process is the same as that of QRD-based squared MMSE method (see section 2.2.3). The complexity results are given in Table 2.9 where N is the antenna number.

Table 2.9 Computational complexity of QRD based square-root MMSE on two platforms

		real multiplication	real addition	square root	real reciprocal
ASICs	$\underline{H} = \underline{Q}\,\underline{R}$	$\frac{16N^3 - 3N^2 + 5N}{3}$	$\frac{25N^3 - 15N^2 - 4N + 3}{3}$	N	N
	$G'_{MMSE} = \frac{1}{\sigma} Q_2 Q_1^H$	$2N^3 + N^2$	$2N^3 - 2N^2$		1
	total	$\frac{22N^3 + 5N}{3}$	$\frac{31N^3 - 21N^2 - 4N + 3}{3}$	N	N+1
RCAs	$\underline{H} = \underline{Q}\,\underline{R}$	$8N^3$	$8N^3 - N^2 - 2N$	N	N
	$G'_{MMSE} = \frac{1}{\sigma} Q_2 Q_1^H$	$2N^3 + 3N^2 + N$	$2N^3 + N$		1
	total	$10N^3 + 3N^2 + N$	$10N^3 - N^2 - N$	N	N+1

2.4 Complexity of different linear MMSE MIMO detection schemes

To choose suitable algorithms for hardware implementation, the complexity of different MIMO detection schemes needs to be compared for RCA and ASIC platforms, respectively. The comparison is performed between QRD-based square-root MMSE scheme and three squared MMSE MIMO detection schemes using different matrix inversion approaches.

2.4.1 System requirement for matrix *R*

Linear MMSE MIMO detectors aim to obtain matrix G_{MMSE} so that the transmitted symbols can be estimated by multiplying the received symbols with G_{MMSE}. However, numerous processing procedures following MIMO detection require matrix R given by QR decomposition. For example, it can be used to calculate squared Euclidean distance (ED) during symbol detection. The complexity involved by ED calculation can be reduced by half because of the property of the upper triangular matrix R [13]. Therefore, the complexity of solving both G_{MMSE} and R should be counted in order to figure out suitable solutions for hardware implementation. Considering that squared MMSE algorithm based on block-wise or DMI inversion method can not provide R, additional QR decomposition is needed for these two squared MMSE schemes after computing G_{MMSE}. In addition, for three squared MMSE schemes, the operation amounts for computing $V^{-1}H^{H}$ need to be counted.

2.4.2 Complexity evaluation on linear MMSE MIMO detection schemes

Table 2.10 gives the total operation numbers of the four schemes on RCA and ASIC platforms, including three squared MMSE schemes with DMI, block-wise and QRD-based matrix inversion method, and one square-root MMSE scheme on the basis of QRD.

Table 2.10 Complexity comparison of MIMO detection schemes on two platforms

scheme		platform	RCA				ASIC			
		antenna number	N=2	N=3	N=4	N=8	N=2	N=3	N=4	N=8
squared MMSE	with analytic/ block-wise inversion	real multiplication	102	372	844	6680	71	285	726	6008
		real addition	83	328	778	6444	55	244	658	5968
		division (1/real)	3	4	6	12	3	4	6	12
		square root	2	3	4	8	2	3	4	8
	with DMI inversion	real multiplication	132	414	944	7104	96	337	812	6612
		real addition	117	378	874	6708	85	310	758	6280
		division (1/real)	4	6	8	16	4	6	8	16
		square root	2	3	4	8	2	3	4	8
	with QRD-based inversion	real multiplication	110	350	804	6120	86	300	720	5820
		real addition	91	307	727	5807	67	257	643	5507
		division (1/real)	2	3	4	8	2	3	4	8
		square root	2	3	4	8	2	3	4	8
square-root MMSE (QRD-based)		real multiplication	94	300	692	5320	62	203	476	3768
		real addition	74	258	620	5048	41	159	401	3489
		division (1/real)	3	4	5	9	3	4	5	9
		square root	2	3	4	8	2	3	4	8

First, an important observation comes from the comparison among different schemes. In terms of addition and multiplication, QRD-based square-root MMSE is the least expensive scheme in terms of computational complexity for given antenna numbers. This is followed by QRD-based squared MMSE and then block-wise squared MMSE. See Fig. 2.2. DMI is the most computationally intensive method, because QR decomposition for solving R contributes a large amount of additional workload.

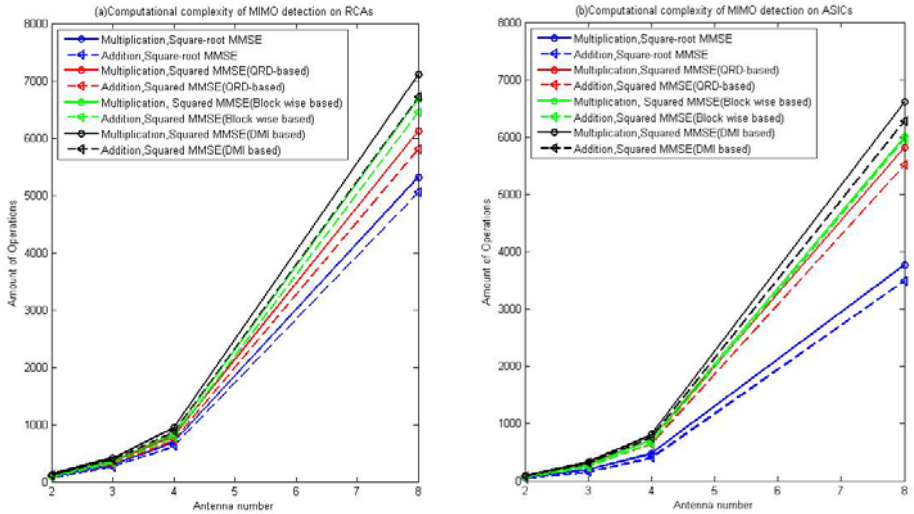

Fig. 2.2 Complexity comparison of MMSE MIMO detection schemes on two platforms (multiplication: solid lines, addition: dashed lines)

Second, Fig. 2.3 shows the operation ratio of the two platforms in terms of multiplication and addition.

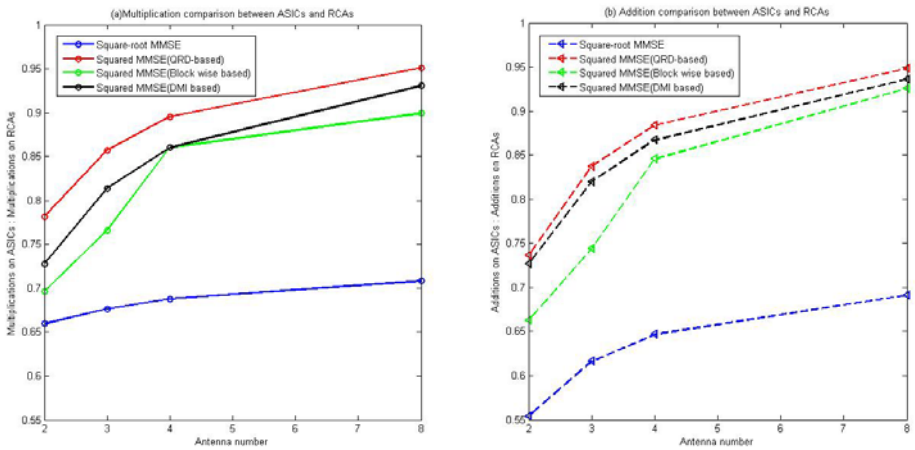

Fig. 2.3 Complexity ratio of MMSE MIMO detection schemes on two platforms

It can be seen that all the ratios are less than 1. This means mapping on ASICs still leads to lower complexity than that on RCAs, since FUs in RCA platforms are developed to map several operations while ASICs are designed to process each specific operation. Similar to the analysis of matrix inversion approaches, the computational complexity of all MIMO detection schemes

increases with antenna numbers for both platforms. However, the differences in complex between two platforms reduce when antenna number increases. It is worth mentioning that the operation ratios dramatically increase by about 0.2 for three squared MMSE schemes when antenna number changes from 2 to 8, while only increase by 0.05 for multiplication and less than 0.15 for addition considering square-root MMSE. This means the advantage of ASICs is more obvious than RCAs to map QRD-base square-root MMSE scheme that is the least computationally demanding scheme. Nevertheless, RCAs have some strength over ASICs. For example, when it is required to deal with different antenna numbers, RCA platform can be configured dynamically to fulfill system requirement. Also, RCAs can be used to map various functions in a time-multiplexing manner. For example, a MMSE estimation algorithm is first mapped on a RCA platform for channel estimation, and after that, its hardware resources are reallocated to perform linear MMSE MIMO detection [2].

To summarize, two QRD-based schemes are the least computationally demanding linear MMSE MIMO detection solutions for the two hardware platforms. Furthermore, the influence of hardware platform on algorithmic complexity decreases as antenna number increases. In addition, ASICs are more suitable to map QRD-base square-root MMSE scheme than RCAs.

CHAPTER 3

3 Accuracy Comparisons of Linear MMSE MIMO Detection Algorithms

Algorithmic accuracy is another significant factor to be considered in additional to the computational complexity discussed in the previous chapter.

In this chapter, the mean squared error (MSE) of G_{MMSE} is used to measure algorithmic accuracy, as defined in (3.1)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{G}_i - G_i)^2, \tag{3.1}$$

where $\hat{G}_i$ is the result of matrix G_{MMSE} calculated using different MIMO detection schemes, G_i is the result of G_{MMSE} obtained from floating-point computations using Matlab inversion function, and N is the total number of G_{MMSE} computations, which is equal to the multiplication of frequency subcarriers number and simulation iteration number. To clearly compare MSEs of various schemes in one figure, only the magnitude of the result is used.

By comparing the mean squared errors (MSEs) of different schemes introduced in Chapter 2, the scheme with the minimum MSE will be considered for hardware implementation. Note that both floating-point and fixed-point accuracy are considered in this chapter.

3.1 Environment setup

This chapter employs an $N \times N$ MIMO system (N= 4 or 8) and simulates using an 802.11n setup. Assume that channel model is 3GPP_EVA_70, frequency subcarrier number is 512 and carrier frequency spacing is 15kHz. Channel coding rate is 1/2 and BCJR algorithm with 6 internal iterations is used for decoding; modulation adopts 64-QAM. The number of simulation iteration is set to be 3000.

3.2 Algorithm enhancement by dynamic scaling

According to the comparison result in Chapter 2, two QRD-based MMSE schemes require the least operation amounts, which are the potential solutions for implementation. However, division and square root are two computationally complex operations involved in QR decomposition, thus how to simplify them is an important problem during hardware design. One solution is to limit input data range by employing dynamic scaling technique.

3.2.1 QRD with dynamic scaling technique

QRD with dynamic scaling technique is shown in Algorithm 3 in which line (2)~(8) are the corresponding dynamic scaling steps [4].

(1) *for i=1 to N*
(2) $x(i) = \|A_i\|^2$
(3) $\alpha = ceil(\log_2(x(i))/2)$ % Scaling factor
(4) $u\_in = x(i) \times 2\text{^}(-2 \times \alpha)$
(5) $u = sqrt(u\_in)$ % $u = sqrt(u\_in) = sqrt(x(i)) \times 2\text{^}(-\alpha)$
(6) $d = 1/u$ % $d = 1/u = 1/(sqrt(x(i)) \times 2\text{^}(-\alpha))$
(7) $R_{ii} = u \times 2\text{^}\alpha$ % $R_{ii} = u \times 2\text{^}\alpha = sqrt(x(i)) = \|A_i\|$
(8) $Q_i = d \times 2\text{^}(-\alpha) \times A_i$ % $Q_i = d \times 2\text{^}(-\alpha) \times A_i = A_i / \|A_i\|$
(9) *for j=i+1 to N*
(10) $R_{ij} = \langle Q_i, A_j \rangle$
(11) $P_{ij} = R_{ij} Q_i$
(12) $A_j = A_j - P_{ij}$
(13) *end for*
(14) *end for*

Algorithm 3: QRD with dynamic scaling

Scaling factor for column norm's square is computed in line (3). In hardware, this factor means a multiple of 2 for shifting operations. Line (4) is used to scale input value of square root operation into the range from 0.25 to 1. Consequently, the square-root result in line (5) is limited to the range of 0.5 and 1. Correspondingly, the divisor in line (6) does not exceed the desired range that is from 0.5 to 1. Diagonal *R* elements and column *Q* are restored by exploiting line (7) and (8), that is to say, the scaling impact on QRD result is eliminated.

3.2.2 Impact of dynamic scaling on algorithm accuracy

Table 3.1 summarizes MSEs of four scenarios for QRD-based squared MMSE and square-root MMSE with / without using dynamic scaling technique for a 4×4 MIMO system. All results are obtained by using floating-point computations.

Table 3.1 MSE comparison in terms of dynamic scaling

	squared MMSE based on QRD	square-root MMSE based on QRD
without dynamic scaling	1.333459e-025	1.130604e-029
with dynamic scaling	1.608063e-025	1.126297e-029

From Table 3.1, it can be seen that dynamic scaling does not bring obvious effects on MSE for the two QRD-based linear MMSE MIMO detection schemes. Hence, dynamic scaling technique can be adopted to save hardware resources.

3.3 Fixed-point word length for QRD-based schemes

This thesis aims to implement a MIMO detection algorithm in 16 bits fixed-point platform. Therefore, besides floating-point accuracy, it has to include the accuracy comparison of two QRD-based schemes with fixed-point format.

To evaluate fixed-point accuracy of the two algorithms, the first step is to find the maximum and minimum value of every intermediate variable and final result so that their fixed-point representation can be determined.

Table 3.2 list recorded data distributions and representation formats of some key variables for the two schemes with dynamic scaling. Q and R are the QRD results given by Algorithm 3 (QRD algorithm with dynamic scaling). Squared norm, scaling factor, *u_in*, and u are variables related to dynamic scaling in line (2) ~ (6) of Algorithm 3, *projection* means the orthogonal projection P_{ij} in line (11), and *internal Q* is the updated vector A_j in line (12). x is an intermediate variable in line (8) of Algorithm 2 (inversion algorithm for upper triangular matrix) required in squared MMSE method only.

For columns of fixed-point format in the tables: totally 16 bits are used to represent every variable; the part before decimal point indicates bit number allocated to both sign bit and integer part, while part after decimal point indicates bit number for fraction part. The corresponding MSEs based on these fixed-point format computations will be presented in section 3.4.

It can be seen from Table 3.2 that, the representation format of most variables are the same for both schemes when comparing a 4×4 MIMO system with an 8×8 system. For squared MMSE scheme, only four variables have different formats, for example x and α. Their bit numbers for fraction parts or bit number for integer are less when N=4. This may lead to a lower accuracy for a 4×4 MIMO system compared with an 8×8 system, because some numbers are too small to be represented with limited fraction bits. For square-root MMSE, majority of variables have the same fixed-point representation format except α.

On the other hand, two schemes can be compared considering the same antenna number. For those variables that both schemes employ, the majority of them have smaller value range when using square-root MMSE so that more bits are allocated for their fraction part. For example, for variable *projection*, 12 bits are assigned to fraction part using squared MMSE when N=4 or 8, whereas 2 more bits are assigned to its fraction part (signed 2.14 format) when using square-root MMSE. This may result in higher accuracy for square-root MMSE than for squared MMSE.

Table 3.2 Key data ranges and fix-point formats of two QRD-based schemes

scheme	variable	N=4			N=8		
		data range		format	data range		format
Squared MMSE	Q	min	-0.979572	signed 1.15	min	-0.904264	signed 1.15
		max	0.998216		max	0.984754	
	R	min	-5.68403	signed 4.12	min	-6.47751	signed 4.12
		max	5.90923		max	7.84291	
	internal Q	min	-2.79354	signed 4.12	min	-3.48027	signed 4.12
		max	4.57374		max	6.56899	
	projection	min	-3.51224	signed 4.12	min	-4.06102	signed 4.12
		max	4.23233		max	4.73996	
	$\|A_i\|^2$	min	1.22e-05	unsigned 6.10	min	5.3145e-05	unsigned 6.10
		max	34.919		max	61.5113	
	α	min	-8	signed integer 5 bits	min	-7	signed integer 4 bits
		max	3		max	3	
	u_in	min	0.25	unsigned 1.15	min	0.25	unsigned 1.15
		max	1		max	1	
	u	min	0.5	unsigned 1.15	min	0.5	unsigned 1.15
		max	1		max	1	
	x	min	-561.826	signed 11.05	min	-265.576	signed 10.06
		max	528.77		max	195.114	
	G_{MMSE}	min	-7.60842	signed 4.12	min	-3.97635	signed 4.12
		max	7.41672		max	4.65013	
square root MMSE	Q	min	-0.987303	signed 1.15	min	-0.931782	signed 1.15
		max	0.99083		max	0.916176	
	R	min	-1.69019	signed 3.13	min	-1.69498	signed 3.13
		max	2.20791		max	2.48328	
	internal Q	min	-1.4064	signed 2.14	min	-1.31563	signed 2.14
		max	1.33951		max	1.35066	
	projection	min	-1.26241	signed 2.14	min	-1.06896	signed 2.14
		max	1.26895		max	1.05804	
	$\|A_i\|^2$	min	3.65e-03	unsigned 3.13	min	8.708e-03	unsigned 3.13
		max	4.87486		max	6.16668	
	α	min	-4	signed integer 4 bits	min	-3	signed integer 3 bits
		max	2		max	2	
	u_in	min	0.25	unsigned 1.15	min	0.25	unsigned 1.15
		max	1		max	1	
	u	min	0.5	unsigned 1.15	min	0.5	unsigned 1.15
		max	1		max	1	
	G_{MMSE}	min	-7.60842	signed 4.12	min	-4.18985	signed 4.12
		max	7.41672		max	4.18585	

However, it is worth noting that when validating fixed-point representation format, some variables of squared MMSE scheme overflow. By contrast, this does not happen with square-root MMSE method. In the following, reasons of the overflow problem and the proposed solution are discussed.

First, the bits assigned to fraction parts of squared norm are not enough and its accuracy can not be guaranteed. This leads to some errors in the following computation steps. In some cases, its value is too small to be represented correctly using assigned bit numbers, which consequently results in wrong scaling factor α. After shifting wrong bits, the inputs of square root and division will have wrong results. Hence, subsequent variables may not be represented by the assigned bits and overflow happens.

Second, the process used to invert upper triangular matrix *R* introduces errors when using fix-point format. Data ranges are quite large for some intermediate variables in this algorithm, such as *x* in Table 3.2. This is also true for the matrix inversion result. This means that some small values will be represented with errors due to the limited precision. Moreover, this is an iterative algorithm and errors will be accumulated. In addition, if *R* cannot be denoted with high accuracy, errors will be propagated to the following computations. To sum up, overflow may happen for intermediate variables involved in this matrix inversion approach, and this may result in overflow in squared MMSE scheme.

Third, overflow occurs most frequently for the last column of matrix *Q*. This reveals that error is accumulated as QRD proceeds. Thus, some intermediate variables such as *projection* and *internal Q* need to use more bits.

To avoid overflow for squared MMSE scheme, bit numbers for fraction part of some variables should be increased, such as squared norm, elements of matrix *R*, and some intermediate variables used for upper triangular matrix inversion and QR decomposition. Table 3.3 lists representation formats which eliminates overflows for 4×4 systems. *y* and *q* are two intermediate variables used in Algorithm 2.

Table 3.3 Adjusted representation formats and MSE for squared MMSE method

	$\|A_i\|^2$	*R*	*internal Q*	*projection*	*x*	*y*	*q*
total bits	27	22	22	22	22	22	22
format	unsigned 6.21	signed 4.18	signed 4.18	signed 4.18	signed 11.11	signed 11.11	signed 12.10

3.4 Fixed-point word length for QRD-based schemes

In this part, MSEs of all four schemes are calculated in floating-point format. Additionally, two QRD-based schemes with the dynamic scaling technique are studied in fixed-point format. Table 3.4 shows all the results considering 4×4 and 8×8 MIMO systems.

Table 3.4 MSEs of different schemes in floating-point/ fixed-point format

scheme		representation format	N=4	N=8
squared MMSE	block-wise inversion	floating point	2.041885e-29	4.22965e-30
	DMI	floating point	3.220027e-28	3.22176e-29
	QRD-based inversion	floating point (without dynamic scaling)	1.333459e-25	1.1204e-26
		floating point (with dynamic scaling)	1.608063e-25	1.29745e-26
		16-bit fixed point without overflow (with dynamic scaling)	1.271787e-02	2.83619e-03
square-root MMSE (QRD-based)		floating point (without dynamic scaling)	1.130604e-29	2.5093e-30
		floating point (with dynamic scaling)	1.126297e-29	2.56651e-30
		16-bit fixed point (with dynamic scaling)	1.216228e-05	5.82651e-06

For floating-point computations, two facts are observed. First, no matter how many antennas a communication system has, QRD-based square-root MMSE provides the highest computation accuracy for MIMO detection. Block-wise squared MMSE is the second best solution with doubled MSE. The MSE of DMI squared MMSE is about ten times higher than that of the block-wise based approach. The worst situation occurs when using QRD-based squared MMSE scheme given that the MSE is 4 orders of magnitude bigger than that of the best solution. Second, based on the comparison between different antenna numbers, the MSE for the scenario of N=4 is about nine times bigger than that of N=8 for each scheme. One possible reason for this experimental result may come from the reference value of G_{MMSE} given by MATLAB function, since the MSEs in this thesis are the mean square error between the result of each scheme and the reference calculated by MATLAB.

Considering the fixed-point format of two QRD-based MMSE schemes, the following conclusion is drawn. For $N \times N$ systems, the squared MMSE approach needs more bits than the square-root MMSE scheme, while still showing a lower computational accuracy. Additionally, the squared MMSE method requires more computation steps than the square-root MMSE approach, including the computation of matrix V and R's inversion. This may result in more computation errors and thus the lower MSE value.

3.5 Simulation of MIMO system performance

Combining the complexity discussion in chapter 2 and the accuracy comparison in this chapter, QRD-based square-root MMSE scheme shows the least computation complexity among four different schemes and higher accuracy in comparison to QRD-based squared MMSE scheme in fixed-point computation. Therefore, QRD-based square-root MMSE scheme is adopted in this work for hardware implementation.

Before implementation, system simulation is performed to assess the performance of the whole MIMO system when employing the adopted detection scheme. In this work, frame error rate (FER) is adopted. Simulation results are shown in Fig. 3.1.

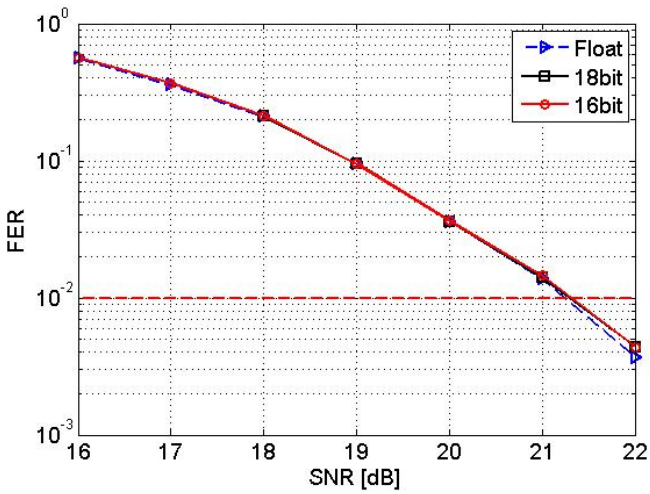

Fig. 3.1 Simulation of system performance

From Fig. 3.1, it can be seen that system performance when using 16-bit fixed-point format is almost the same as those using 18-bit fixed point format. Moreover, floating-point format does not show apparent advantage over two fixed-point formats until SNR goes above 22dB. Therefore, 16-bit fixed-point format is used to implement the adopted QRD-based square-root MMSE scheme.

CHAPTER 4

4 Hardware Implementations for Fixed-point Square-root MMSE Algorithm

In this chapter, three architectures of QRD-based square-root MMSE MIMO detection algorithm are developed for 4×4 MIMO systems using standard digital cell libraries and are verified using an FPGA platform. The three architectures include two pipeline versions and a time-multiplexed scheme. Performance of these three implementations with respect to throughput, speed and resource utilization, are summarized.

4.1 Description of main building blocks

According to the QRD-based square-root MMSE scheme described in Chapter 2, an augmented channel matrix $\underline{H}$ is constructed to calculate G_{MMSE} by extending matrix H. In this chapter $AugH$ is used to denote the matrix of augmented $\underline{H}$. Recall that the modified Gram-Schmidt QRD (Algorithm 1) is an iterative algorithm and consists of two processes: a) the outer loop is used to obtain norm (R_{ii}) and normalization vector (Q_i) of $AugH_i$; b) the inner loop is to update columns $AugH_j$ ($j=i+1$ to N) by first calculating inner-product (R_{ij}) of vector $AugH_j$ and Q_i, and then updating $AugH_j$. Thus, two components are designed. One is used to receive column vector $AugH_i$ and produce the corresponding R_{ii} and Q_i, called $Q_i\_R_{ii}$ *Component* ($Q_i\_R_{ii}$ *C*). The other is to calculate R_{ij} and update augmented $AugH_j$, called $R_{ij}\_H_jUpd$ *Component* ($R_{ij}\_H_jUpd$ *C*). In the following, the two blocks are described in details and their device utilization after place and route (PNR) are shown.

4.1.1 $Q_i\_R_{ii}$ Component

The block diagram of $Q_i\_R_{ii}$ *Component* is shown in Fig. 4.1 with five sub-blocks and two delay elements.

1. Norm_alfa module

This module is developed to calculate scaling factor and the scaled norm-square for vector $AugH_i$ using dynamic scaling technique.

The non-scaled norm-square is calculated by multiplications of vector elements and additions of multiplication results. Note that fixed-point implementation requires adjustments of operation's results according to the word length given in Table 3.2, and the word length adjustments are performed throughout the entire hardware design.

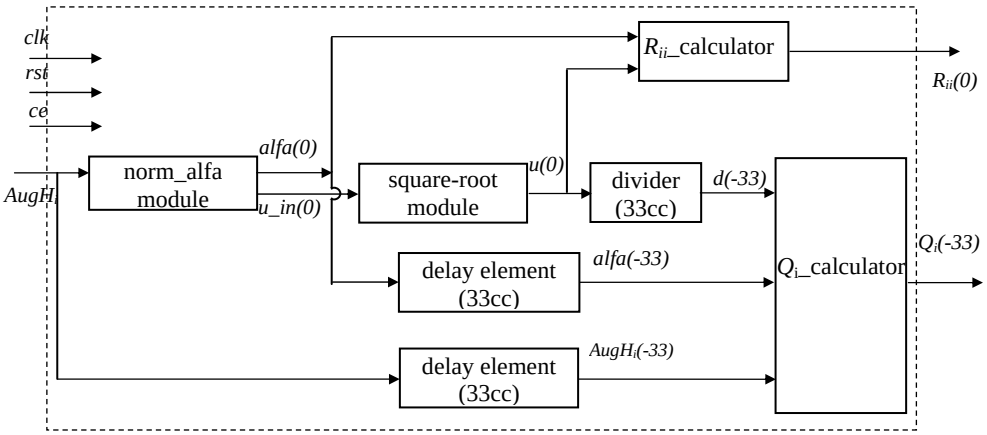

Fig. 4.1 Block diagram of Q_i R_{ii} *Component*

Next, the norm-square for vector $AugH_i$ is scaled to the range between 0.25 and 1, denoted as *u_in*. Scaling factor α is found based on squared norm. It is the bit location where the first '1' appears in binary squared norm. Then, scaling can be done by shifting the squared norm to the right 2α bits according to line (4) in Algorithm 3. After that, result adjustment has to be done because *u_in* has different fixed-point format with the scaled norm-square. The latter is represented by unsigned 3.13 format, whereas *u_in* is by unsigned 1.15. This means, the result should be further shifted two bits to the left to get the correct representation of *u_in.* The relationship between α and shift operation is shown in Table 4.1.

Table 4.1 Shift operation to scale norm-square

α	shift direction	shifting bit
-4	left	10
-3	left	8
-2	left	6
-1	left	4
0	left	2
1	No shift	0
2	right	2

2. Square-root module

This module receives scaled norm-square named *u_in* in Fig. 4.1 and send out the vector norm *u*. Xilinx IP Core Generator is used to implement this module. Since square root is a compute-intensive operation and this module is part of critical path of the entire system, its performance is a bottleneck of system's frequency. As

synthesis report shows, the minimum period of square-root module is 28.913ns including 33.3% logic and 66.7% route; and the device usage of this block after PNR is shown in Table 4.2.

Table 4.2 Resource utilization of square-root module

slice logic utilization	used
number of slice registers	0
number of slice LUTs	248
number of occupied slices	78
number of LUT flip flop pairs used	248
number of OLOGICE1/OSERDESE1s	14

3. R_{ii}_calcultor

It is designed to eliminate the impact of the dynamic scaling on R_{ii}. According to line (7) in Algorithm 3, R can be restored by shifting u to the right α positions. However, similar to the scaling of norm-square, u should be shifted to the right 2 more bits due to different fixed-point formats of R and u. The shifting operation is concluded in Table 4.3.

Table 4.3 Shift operation to restore R_{ii}

α	shifting direction	shifting bit
-4	right	6
-3	right	5
-2	right	4
-1	right	3
0	right	2
1	right	1
2	No shift	0

4. Divider

The divider is used to compute the reciprocal of a vector norm and is implemented by Xilinx IP Core Generator. Unsigned operation is selected, since norm is a positive real number. Regarding the word-length, the fractional is set to be 29 bits according to Table 3.2. The adopted divider is fully pipelined, resulting in a throughput of 1 division per clock cycle. Note that the latency of the generated divider is decided by

$$Latency = quotient's\ word\ length + fractional's\ word\ length + 2. \quad (4.1)$$

In this work, quotient and dividend have the same wordlength and the latency is 33 clock cycles (cc). Divider's maximum frequency is 515.464MHz as synthesis report shows. Table 4.4 summarizes PNR results of the adopted divider.

Table 4.4 Divider's device utilization

slice logic utilization	used
number of slice registers	1145
number of slice LUTs	620
number of occupied slices	290
number of LUT flip flop pairs used	895

5. Q_i_calculator

In this module, $AugH_i$ is multiplied with the reciprocal of its norm. 16-bit Q_i is obtained by truncating the 32-bit multiplication result in accordance to α, see Table 4.5.

Table 4.5 Truncating operation to restore Q_i

α	MSB	LSB
-4	24	9
-3	25	10
-2	26	11
-1	27	12
0	28	13
1	29	14
2	30	15

6. Two 33cc delay elements

Because the divisor has a 33 cc latency, the corresponding $AugH_i$ and α need to be delayed for 33cc to produce correct Q_i. Thus, one delay element is inserted on the $AugH_i$ to Q_i*_calculator* path and the other on the α to Q_i*_calculator* path.

Because of the involved divider and multipliers, Q_i_R_{ii} *Component* consumes a large amount of hardware resources, see Table 4.6.

Table 4.6 Device utilization comparison of Q_i_R_{ii} *Component*

slice logic utilization	Q_i_R_{ii} Component
number of slice registers	1695
number of slice LUTs	2033
number of occupied slices	785
number of LUT flip flop pairs used	2306
DSP48E1s	32

4.1.2 R_{ij}_H_j*Upd* Component

Fig. 4.2 shows the block diagram of R_{ij}_H_j*Upd Component*.

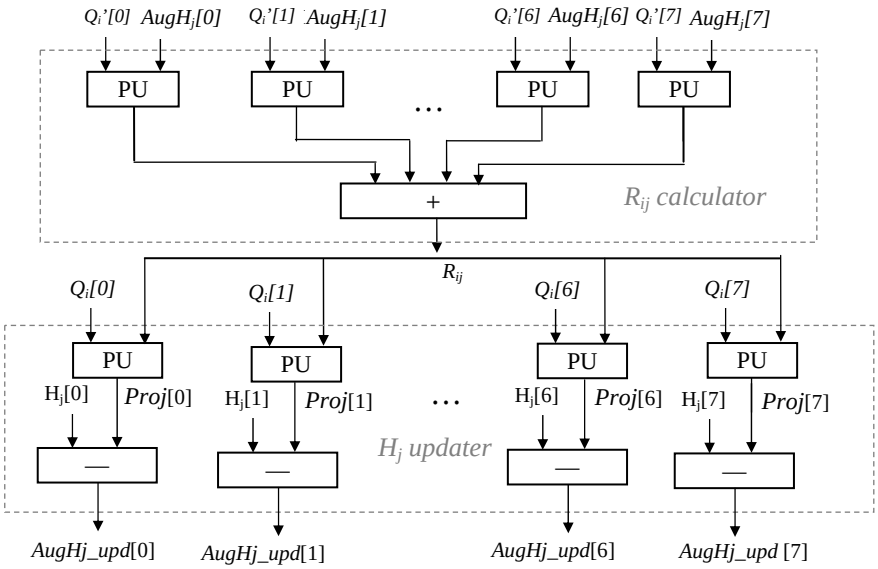

Fig. 4.2 Block diagram of R_{ii} H_i*Upd Component*

Processing units (PUs) are used to calculate R_{ij} and the projection of $AugH_j$ on Q_i. A PU is developed for multiplication of two 16-bit complex-valued numbers. In each PU, there are four 16-bit real-valued multipliers and two 32-bits real-valued adders/ subtracters.

Table 4.7 lists the hardware consumption of this component. It can be clear seen that the hardware consumption of this component is about 9 times less than that of $Q_i\_R_{ii}$ *Component*. To reduce hardware cost, the number of the $Q_i\_R_{ii}$ *Components* adopted in system architecture should be as small as possible.

Table 4.7 Device utilization comparison of $R_{ij}\_H_j$*Up Component*

slice logic utilization	$R_{ij}\_H_j$*Upd* Component
number of slice registers	116
number of slice LUTs	196
number of occupied slices	52
number of LUT flip flop pairs used	196
DSP48E1s	32

4.2 The first pipeline scheme

4.2.1 Architecture description

The first pipeline architecture is shown in Fig. 4.3, stage one is a matrix extending module (*matr_exd*) which is designed to build augmented matrix *AugH* from input channel matrix *H*.

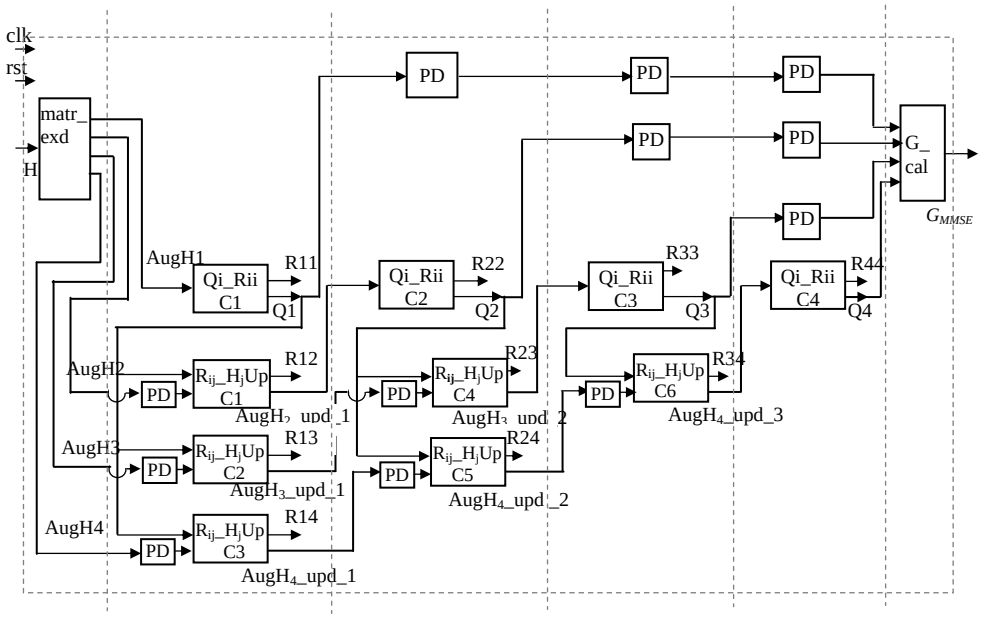

Fig. 4.3 Architecture of the first pipeline version

The following four stages are used to implement QR decomposition with each stage processing one column. First, in stage two, one *Q_i_R_{ii} Component* is employed to calculate Q_1 and three *R_{ij}_H_jUpd Components* are used to calculate R_{1j} and update AugH_j for the first time (j=2,3,4). Similarly, stage three consists of one *Q_i_R_{ii} Component* and two *R_{ij}_H_jUpd Components*. *Q_i_R_{ii} C2* receives *$AugH_2$_upd_1st* and computes Q_2; $AugH_3$ and $AugH_4$ are updated for the second time in *R_{ij}_H_jUpd C4* and *R_{ij}_H_jUpd C5*. Finally, in stage 5, only one *R_{ij}_H_jUpd C* is used for Q_4's calculation.

It is worth noting that the input $AugH_j$ / $AugH_j$_upd of every *R_{ij}_H_jUpd Component* has to be delayed 33cc due to the latency of calculating the corresponding Q_i. Thus, delay elements (DEs) are inserted by connecting 33 DFFs in series. Besides, DEs are also used as pipeline registers, denoted as PDs in Fig. 4.3.

The last stage, stage 6, is used to calculate G_{MMSE} matrix when Q_4 is produced. Since it costs 33cc to produce each Q_i, there are three PDEs for Q_1, two for Q_2, and one for Q_3.

4.2.2 Implementation and simulation

1. Behavioral simulation

The processor is designed to compute three matrices in parallel: R, Q, and G_{MMSE}.

The total bit number is 1792 including 256 bits for matrix *R*, 1024 bits for matrix *Q* and 512 bits for matrix G_{MMSE}. The behavioral simulation result is given in Fig. 4.4.

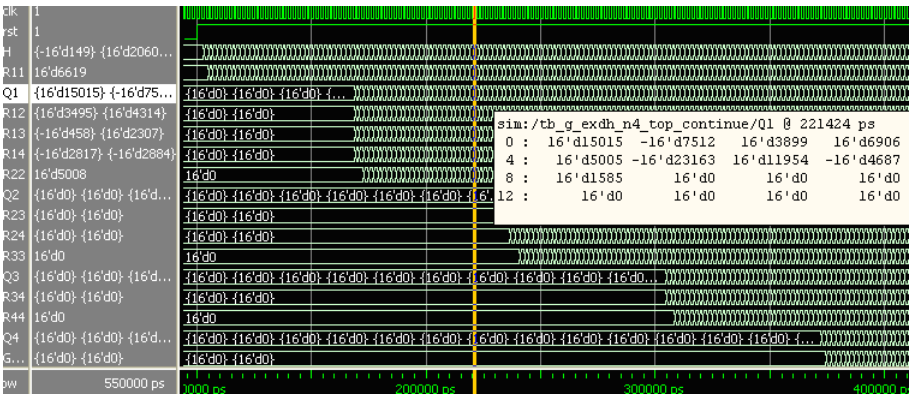

Fig. 4.4 Beheviour simulation of the first pipeline version

Form Fig. 4.4, it can be seen that this pipeline version is able to produce one *R* matrix, one *Q* matrix, and one G_{MMSE} matrix in each clock cycle.

2. Synthesis and PNR

During synthesis and PNR, because three parallel output matrices and the input matrix H have 2304 bits in total, these matrices are not directly connected to IOs. Instead, some ROMs are designed to store input matrices and the corresponding reference model. On the output, a “diff” signal is used to indicate if the results are correct by comparing with the reference output. If they are the same, diff is set ‘0’. This signal is generated at the same clock cycle as matrix G_{MMSE} and is registered before output.

The synthesis report shows that the minimum clock period of the design is 63.556ns (maximum frequency: 15.734MHz). Device utilizations after PNR are provided in Table 4.9 in section 4.5. Fig. 4.5 shows the post PNR simulation using minimum period.

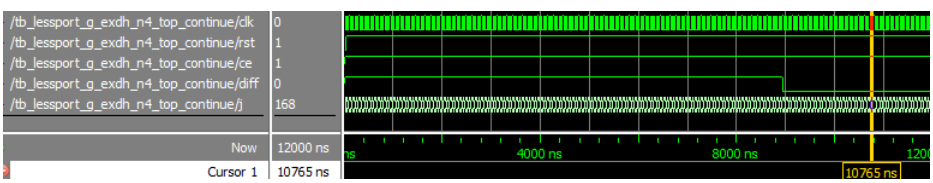

Fig. 4.5 Post PNR simulation for the first pipeline scheme

In this figure, “diff” signal becomes zero when the 139rd clock cycle arrives after reset. This is because it spends 1cc to obtain *AugH* matrix from channel

matrix H, then 33cc to produce each Q_i (i=1,2,3,4) and 1cc to store each Q_i in DFF, 1cc to store G_{MMSE} in DFF after Q_4, and 1cc to store diff in DFF for registered output.

4.3 Time-multiplexed scheme

It is well-known that time-multiplexed architecture is efficient in term of hardware resource but has long computation time because several algorithm operations share the same hardware. In this part, a time-multiplexed scheme is developed to implement QRD-based square-root MMSE MIMO detection. In order to save hardware area, this scheme adopts only one $Q_i\_R_{ii}$ *Component* and one $R_{ij}\_H_j$*Upd Component*.

4.3.1 Architecture description

For every column vector $AugH_j$, QR decomposition steps are similar, i.e., they all employ one $Q_i\_R_{ii}$ *Component* and *(N-j)* $R_{ij}\_H_j$*Upd Components* as designed using the pipeline architecture. As a result, one $Q_i\_R_{ii}$ *Component* and one $R_{ij}\_H_j$*Upd Component* can be shared over time in time-multiplexed scheme. In addition, as described before, some signals need to wait for 33cc due to the latency of divider, therefore, holding element (HE) is introduced to keep signals for some time. As Fig. 4.6 shows, a HE uses a control signal to select data from either input or output's feedback. Different from the pipeline-based delay element in the first pipeline scheme, the HE saves hardware resources especially when the holding period is long.

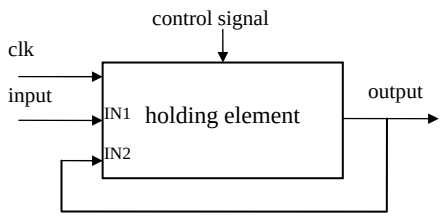

Fig. 4.6 Block diagram of holding element

The block diagram of time-multiplexed architecture is shown in Fig. 4.7: $Q_i\_R_{ii}$ *Component* and $R_{ij}\_H_j$*Upd Component* are indicated in light grey, and HEs are specified in darker grey. A control module shown in the upper part manages all other modules by using select signals.

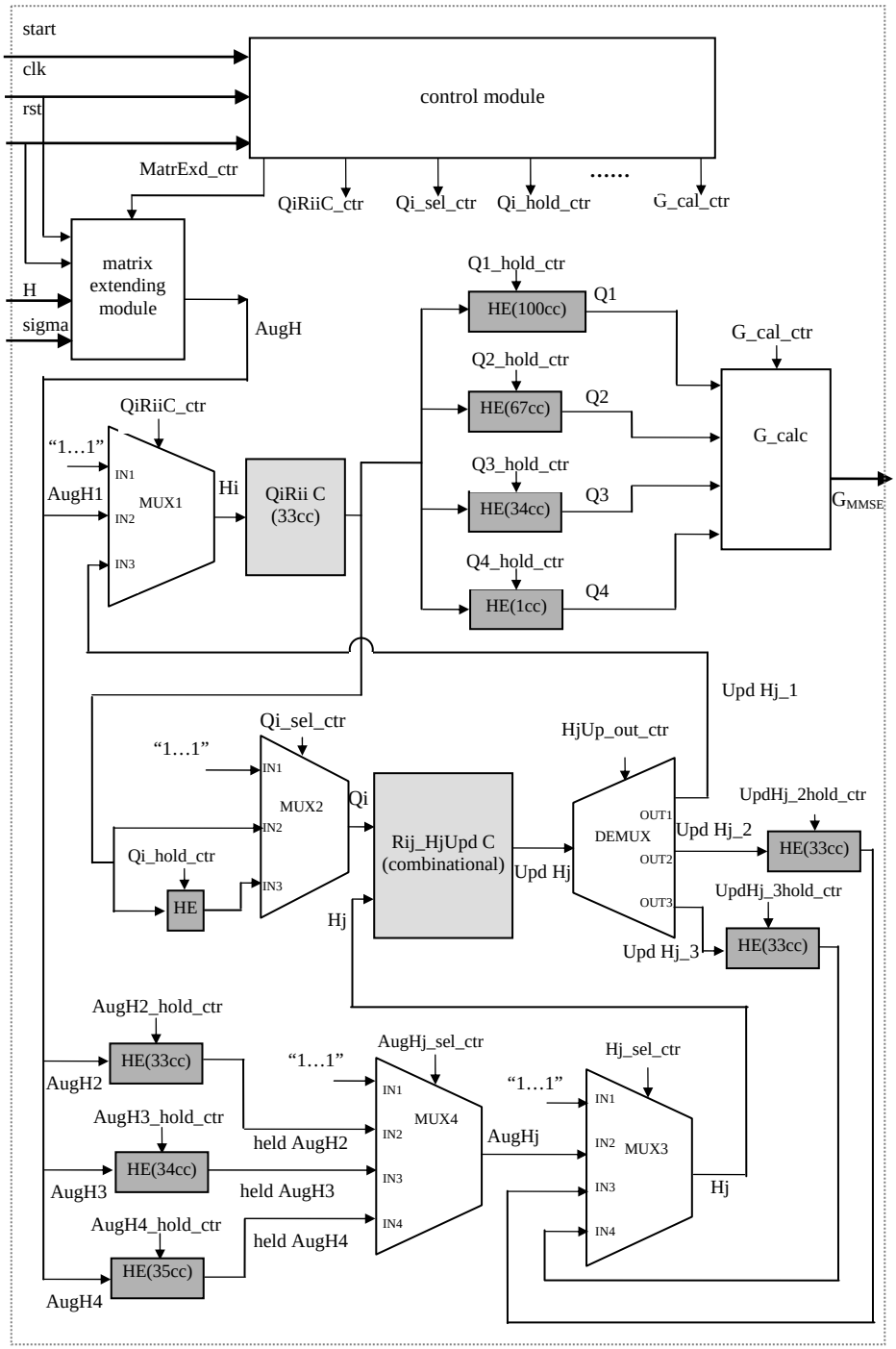

Fig. 4.7 Architecture of time-multiplexed scheme

1. Matrix extending module

Once the system is reset and a start signal arrives, control module gives out a MatrExd_ctr signal to activate this module. Four column vectors of augmented H matrix are sent out in the following clock cycle. $AugH_1$ is connected to MUX1 preparing to calculate Q_1, while $AugH_2$, $AugH_3$ and $AugH_4$ are sent into three 33cc hold elements waiting for Q_1 to calculate R_{ij} and update themselves for the first time. Since the $R_{ij}\_H_jUpd$ *Component* is time-multiplexed, the $AugH_2$ is the first one to enter $R_{ij}\_H_jUpd$ *Component*. After 1cc, its output, the updated $AugH_2$, is sent into $Q_i\_R_{ii}$ *Component* for *Q2*'s calculation. In the meanwhile, $AugH_3$ occupies $R_{ij}\_H_jUpd$ *Component*, therefore $AugH_3$ needs to wait for 34cc since it is given out by the matrix extending module. Similarly, $AugH_4$ needs to wait for 35cc before $R_{ij}\_H_jUpd$ *Component* processes it.

2. $Q_i\_R_{ii}$ *Component*

Iits input H_i has three sources selected by Q_iR_{ii} *C_ctr* signal. The first one is $AugH_1$ when computing Q_1. If the clock cycle is denoted as 1cc when *AugH* is output from matrix extending module, then the MUX1 selects $AugH_1$ at 1cc with Q_1 given out at 34cc. Its second potential input comes from one of $R_{ij}\_H_jUpd$ *Component* outputs when computing Q_2, Q_3 and Q_4. Specifically, MUX1 selects $AugH_{2\_}$ upd_1^{st} to compute Q_2 at 34cc after Q_1 is sent out. This is because $AugH_2$ has to subtract its orthogonal projection on Q_1 to obtain $AugH_{2\_}$ upd_1^{st}. Similarly, the updated $AugH_3$ is allowed to enter $Q_i\_R_{ii}$ *Component* at 67cc when Q_2 is computed, since $AugH_3\_upd\_2^{nd}$ is computed by subtracting orthogonal projection of $AugH_3\_upd\_1^{st}$ on Q_2. $AugH_4\_upd\_3^{rd}$ is selected at 100cc once Q_3 is obtained. Finally, the third input source "1…1" is effective for 128 cc out of the 132cc before next channel matrix comes into system.

The output of this module Q_i has two destinations. One is to calculate G_{MMSE}. As seen in upper right part of Fig. 4.7, Q_1, Q_2, Q_3 and Q_4 need to be held for different time since they are produced at different clock cycles. Another is going to $R_{ij}\_H_jUpd$ *Component* for orthogonal operation. Since this component is time-multiplexed, Q_1 should be held for at least 3cc.The 1st cc is used to calculate R_{12} and update $AugH_2$, the 2nd cc to calculate R_{13} and update $AugH_3$, and the 3rd cc for R_{14} and $AugH_4$. Similarly, Q_2 is held for at least 2cc and Q_3 for 1cc.

3. $R_{ij}\_H_jUpd$ *Component*

As described above, input Q_i (j=1, 2, and 3) need to be held for 3cc, 2cc and 1cc respectively. "1…1" is connected for the rest of the time within a processing iteration for each channel matrix, since it has no work until the divider gives out a new Q_i. This input selection is achieved by combining the HE and MUX2 with $Q_i\_sel\_ctr$ control signal. Another input of this component is H_j. It could be one of $AugH_2$, $AugH_3$, and $AugH_4$ when orthogonalized with Q_1, which is selected by MUX3 and MUX4. It could also come from outputs of this component itself, i.e.,

updated $AugH_3$ and $AugH_4$. For example, when computing $AugH_3\_upd\_2^{nd}$, Hj comes from $AugH_3\_Upd\_1^{st}$. For the rest of the time, it is connected to "1…1".

Next, the output which is updated H_j will go to three directions. Three computation results go to $Q_i\_R_{ii}$ *Component* immediately for Q_i calculation, which are $AugH_2\_Upd\_1^{st}$, $AugH_3\_Upd\_2^{nd}$ and $AugH_4\_Upd\_3^{rd}$. Then, $AugH_3\_Upd\_1^{st}$, $AugH_4\_Upd\_1^{st}$ and $AugH_4\_Upd\_2^{nd}$ should go back to this component for further updating $AugH_j$ (j=3 and 4). Because they need to wait for 33cc for the corresponding Q_i for orthogonalization, some HEs should be inserted. Among them, $AugH_3\_Upd\_1^{st}$ and $AugH_4\_Upd\_2^{nd}$ share one HE while $AugH_4\_Upd\_1^{st}$ will go through another HE, since $AugH_4\_Upd\_1^{st}$ is calculated 1cc later than $AugH_3\_Upd\_1^{st}$ and both need to be kept for 33cc. Therefore, three feedback results are assigned to two signals and enter two HEs, see Fig. 4.7.

4.3.2 FSM design

Based on the description of hardware architecture and time sequence, a finite-state machine (FSM) of this scheme is designed, see Fig. 4.8.

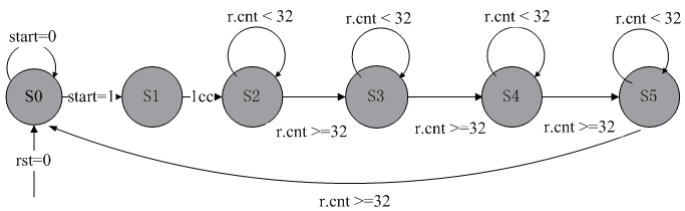

Fig. 4.8 FSM of time-multiplexed scheme

- S0: A ready-for-work state once reset signal is effective. When start signal is high, the system begins to work. After an iteration finishes and system re-enters this state, a control signal is given out to trigger the *G_cal module* for G_{MMSE} calculation of last channel matrix.
- S1: The control module gives *matrix extending module* an enable signal, and *matrix extending module* immediately starts to extend an H matrix to an augmented one. Meanwhile, *G_cal module* outputs G_{MMSE} result for the last channel matrix.
- S2: $Q_i\_R_{ii}$ *Component* receives $AugH_1$ and works for 33cc to calculate Q_1.
- S3: $R_{ij}\_H_jUpd$ *Component* calculates R_{1j} and updated $AugH_j$ (j=2, 3, 4) in the first three continuous clock cycles. $Q_i\_R_{ii}$ *Component* takes 33cc to compute Q_2.
- S4: Similar to S3, $R_{ij}\_H_jUpd$ *Component* works for 2 clock cycles for R_{2j} and updated $AugH_j$ (j=3 and 4). Q_3 is calculated after 33cc by $Q_i\_R_{ii}$ *Component*.

- S5: $R_{ij}\_H_jUpd$ *Component* produces R_{34} and $AugH_4\_Upd\_3^{rd}$ in the first clock cycle. Once Q_4 is computed by $Q_i\_R_{ii}$ *Component* after 33cc, the next state will be S0 to process a new channel matrix.

4.3.3 Implementation and simulation

The synthesis report shows that the minimum period of the time-multiplexed scheme is 96.030ns, i.e. the maximum frequency is 10.413MHz. After PNR, the hardware utilization is reported and is listed in Table 4.9 in section 4.5.

Similar to the pipeline version, a "diff" signal is introduced to indicate computational correctness. Fig. 4.9(a) shows the behavioral simulation result in order to demonstrate time sequence and Fig. 4.9(b) is the post-PNR simulation result.

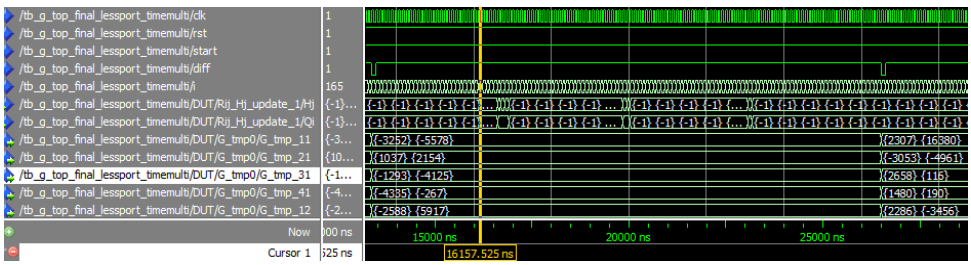

Fig. 4.9(a) Behaviral simulation of time-multiplexed scheme

Fig. 4.9(a) illustrates the process procedure of the first two channel matrices after start signal. It can be seen that the time-multiplexed system outputs a G_{MMSE} matrix every 134cc and diff is kept '0' for 1cc.

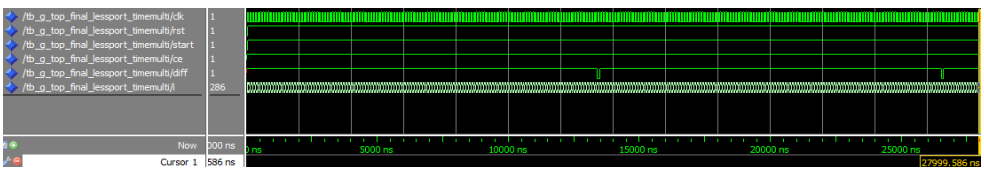

Fig. 4.9(b) Post-PNR simulation of time-multiplexed scheme

Post_PNR simulation result is shown in Fig. 4.9(b) when using the maximum frequency. That "diff" signal becomes zero every 134cc demonstrates that the post-PNR system works correctly.

4.4 The second pipeline scheme

To improve the previous pipeline and the time-multiplexed architectures, a second pipeline scheme is designed. Because the processing procedure for every column is similar, hardware resources for column processing can be shared over time.

Meanwhile, this shared hardware is fully pipeline in order to process one column in each clock cycle.

4.4.1 Architecture description

The idea is to use one *Q_i_R_{ii} Component* and three *R_{ij}_H_jUpd Components* in this pipeline scheme. The block diagram is shown in Fig. 4.10. In this figure, *Q_i_R_{ii} Component* and *R_{ij}_H_jUpd Components* are indicated with light grey blocks.

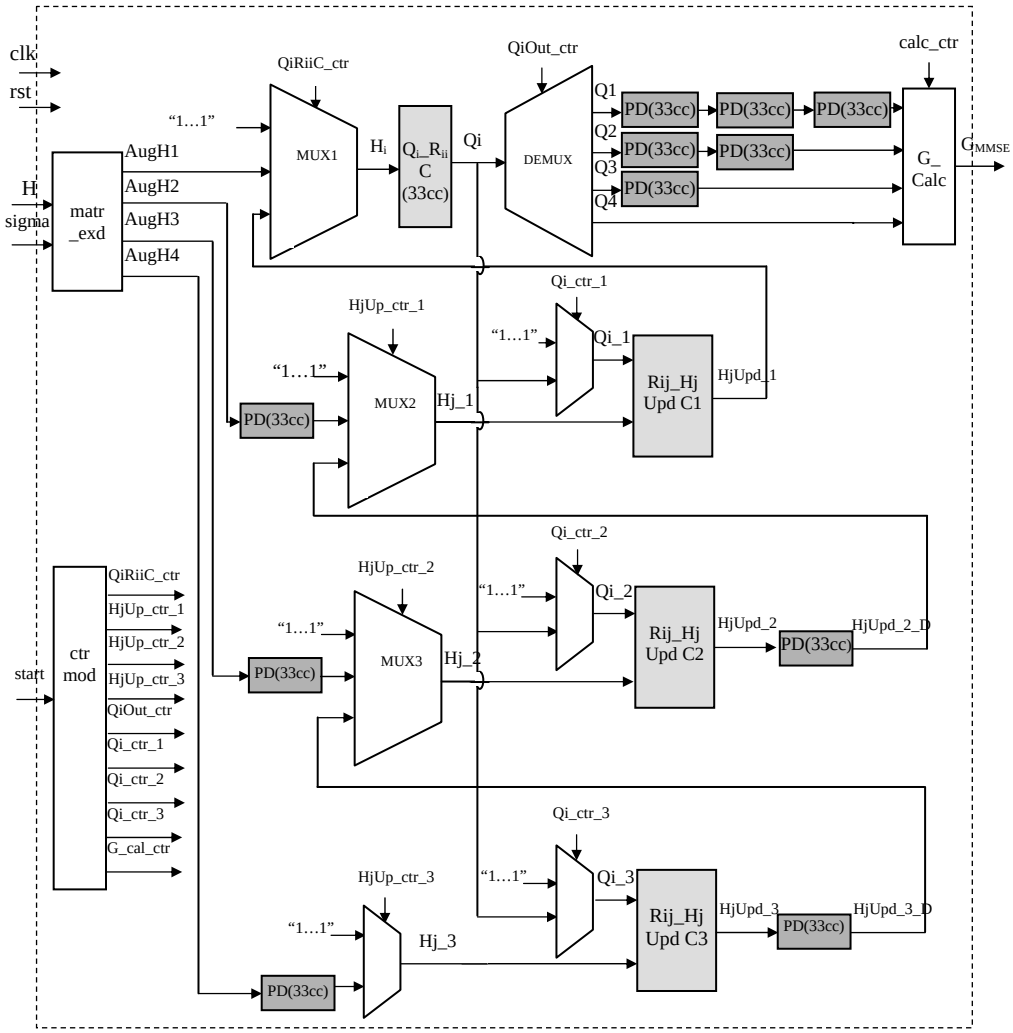

Fig. 4.10 Architecture of the second pipeline scheme

1. $Q_i\_R_{ii}$ *Component*

Its input H_i can come either from $AugH_1$ when processing the first column or from the output of $R_{ij}\_H_jUpd\ C1$ which is the updated H_j vector when processing the 2nd, 3rd and 4th columns. This is achieved by MUX1 with the selecting signal "Q_iR_{ii} *C_ctr*" managed by a control module. This component is the same as $Q_i\_R_{ii}$ *Component* in the 1st pipeline scheme.

2. Three $R_{ij}\_H_jUpd$ *Components*

After Q_1 is calculated, all the three $R_{ij}\_H_jUpd$ *Components* have to work since $AugH_2$, $AugH_3$ and $AugH_4$ need to get updated. When Q_2 is calculated, $AugH_3$ is updated for the second time in $R_{ij}\_H_jUpd\ C1$ and $AugH_4$ is updated in $R_{ij}\_H_jUpd\ C2$. When Q_3 is obtained, only $R_{ij}\_H_jUpd\ C1$ is used to update $AugH_3$ for the 3rd time.

It can be seen that $R_{ij}\_H_jUpd\ C3$ works the least time during one MIMO detection process. Its input H_j is set to "1…1" when the module is idle. This is managed by a multiplexer with a select signal "$H_jUp\_ctr\_3$". Similarly, the input of $R_{ij}\_H_jUpd\ C2$ comes either from $AugH_3$ or from $AugH_4\_Upd\_1^{st}$ that is the output of $R_{ij}\_H_jUpd\ C3$, and for the rest of the time, the input is connected to "1…1". $R_{ij}\_H_jUpd\ C1$ receive its H_j from "1…1", $AugH_1$, or the output of $R_{ij}\_H_jUpd\ C2$ which could be $AugH_2\_Upd\_1^{st}$, $AugH_3\_Upd\_2^{nd}$ and $AugH_4\_Upd\_3^{rd}$.

Because Q_1 takes 33cc to compute, $AugH_2$, $AugH_3$ and $AugH_4$ need to wait for 33cc to compute the correct R_{1j} and updated H_j. This is implemented by pipeline-based delay elements (PDs) in order to process one column in each clock cycle. Once the output of $R_{ij}\_H_jUpd\ C1$ is calculated, it should enter $Q_i\_R_{ii}$ *Component* immediately without any delay, whereas the outputs of both $R_{ij}\_H_jUpd\ C2$ and $R_{ij}\_H_jUpd\ C3$ have to pass through PDEs since they are supposed to wait for the corresponding Q_i.

Like H_j, another input signal Q_i needs to be selected from "1…1" when the component is idle or from the output of $Q_i\_R_{ii}$ *Component* when it is active. Given that these three $R_{ij}\_H_jUpd$ components work at different time, three multiplexers are assigned to each of them with control signals $Q_i\_ctr\_1$, $Q_i\_ctr\_2$, and $Q_i\_ctr\_3$.

3. G_{MMSE} calculator

Q_{i+1} is generated 33cc after Q_i, and it won't get the correct G_{MMSE} until Q_4 is obtained. Therefore, 33cc PDEs are needed to meet the timing requirement, as shown in dark grey at upper right corner of Fig. 4.10.

4. Delay elements

In this scheme, the delay element is designed in the same way as the first pipeline

scheme, i.e., composed of 33 concatenated DFFs, denoted as PD in the architecture diagram.

4.4.2 Timing discussion

The most complicated task of this scheme is to design its timing. Here, the hardware is supposed to be able to output G_{MMSE} in a pipeline manner.

1. Direct design method

The direct design idea is to input a channel matrix H every clock cycle. But this will lead to input conflict for $Q_i\_R_{ii}$ *Component*. Fig. 4.11 explains this problem, in which only some related modules and some intermediate variables' timing are shown.

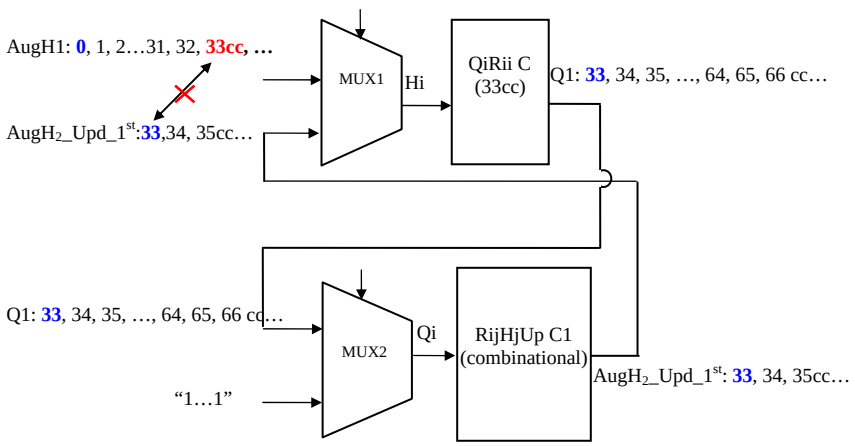

Fig. 4.11 Input confliction using original direct design idea

For example, assume the first channel matrix is input to $Q_i\_R_{ii}$ *Component* at k=0 cc. Note that the arriving time of all signals related to this input are indicated in blue. For this channel matrix, Q_1 will be produced at 33cc and sent to $R_{ij}\_H_jUpd$ *Component* immediately. Because it is a combinational logic circuit, $AugH_2\_Upd\_1^{st}$ is output and delivered to $Q_i\_R_{ii}$ *Component* at the same clock cycle, i.e., 33cc. However, we input one channel matrix every clock cycle, which means a new $AugH_1$ needs to occupy $Q_i\_R_{ii}$ *Component* at 33cc too, and this is indicated in red in Fig. 4.11. Thus, input confliction happens in $Q_i\_R_{ii}$ *Component* from 33cc. As a result, this design fails to realize the required function.

2. Two alternative solutions

To solve the confliction problem, another $Q_i\_R_{ii}$ *Component* could be added into system. However, as pointed out in section 4.1, this component costs a large amount of device resources, so that system hardware area will increase.

Another solution is to delay the input of the 34th channel matrix till 132cc when the first 33 channel matrices have been completely processed. That is, every 33 channel matrices are regarded as a group. This group of matrices enters into the system continuously from *k*=0cc to 32cc with one matrix input per clock cycle. After that, matrix input stops for 99cc. The input of the next group starts from 132cc.

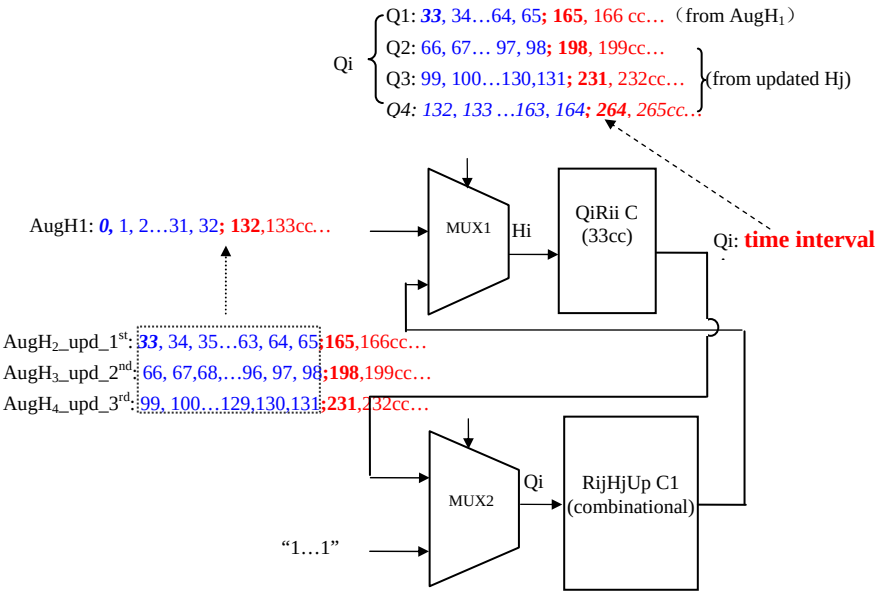

Fig. 4.12 Solution of input rescheduling

In Fig. 4.12, processing of the first group of channel matrices is illustrated in blue color, and the second group is shown in red color. It can be seen that this solution eliminates input confliction in the $Q_i\_R_{ii}$ *Component*. Notice that this kind of input scheduling results in interruptions on data output. As Fig. 4.12 shows, there is a time interval of 100cc in between two consecutive groups. We consider this as a burst input scheduling. To obtain a smooth output, a new input scheduling is introduced as follows.

3. Final design implementation

From the analysis in the previous section, it can be seen that an input group of 33 channel matrices can be processed in a period of 132cc. This means that the architecture can deal with one channel matrix every 4cc. Thus, we can employ a new input scheduling: for each successive 4 clock cycles, input a channel matrix in the first clock cycle and then stop input for the next 3cc. By doing this, matrix output is also generated on every 4cc. The work principle is shown in Fig. 4.13.

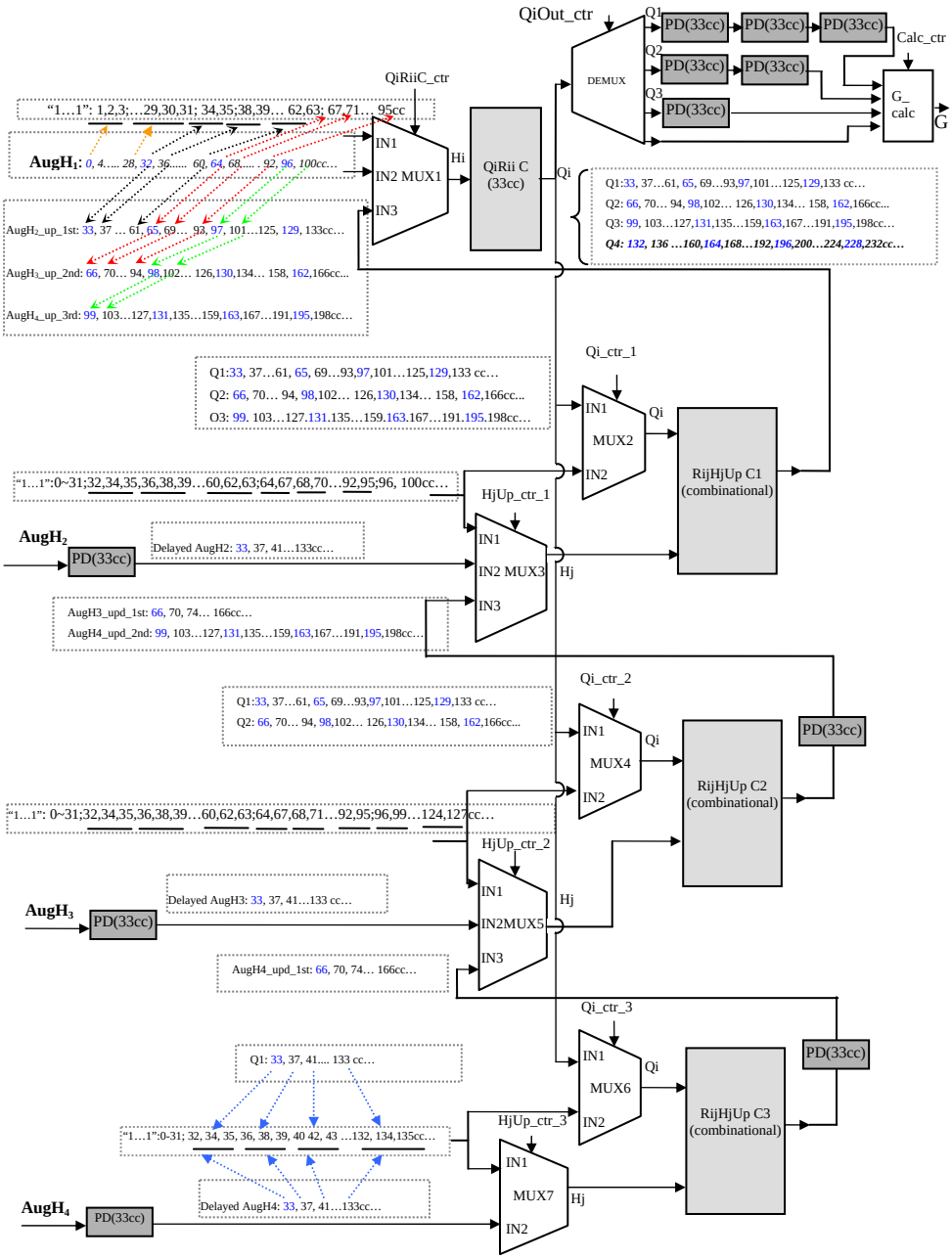

Fig. 4.13 Work principle when adopting new input

First, for $Q_i\_R_{ii}$ *Component*, MUX1 with the select signal Q_iR_{ii} *C_Ctr* is used to select H_i for this component to calculate Q_i. In the upper left corner of this figure, $AugH_j$ (j=1, 2, 3, 4) shown in bold font are sent into the system on every 4cc, i.e. 0, 4, 8, 12…. Orange arrows indicate MUX1's settings in the first 32cc period (from 0 to 31cc). During the second period of 32cc, since $AugH_2\_Upd\_1^{st}$ is computed, the input selection of MUX1 changes as shown in black arrows. In the third period of 32cc (red arrows), MUX1 selects its IN3 port twice on every 4cc for inputing $AugH_3\_Upd\_2^{nd}$ and $AugH_2\_Upd\_1^{st}$. Explained using green arrows, $AugH_4\_Upd\_3^{rd}$ begin to enter $Q_i\_R_{ii}$ *C* to compute Q_4 during the 4th period of 33cc. From this period, $Q_i\_R_{ii}$ *Component* begins to operate with full load and is able to output one Q_i in each clock cycle. The corresponding setting of MUX1 is summarized in Table 4.8(a).

Second, for each of three $R_{ij}\_H_jUpd$ *Components*, there are 2 multiplexers to control its two inputs respectively. $R_{ij}\_H_jUpd$ *C*3 only receives $AugH_4$ to do orthogonalization with Q_1. Thus, it does not work in the first 32cc because no Q_1 is produced yet. Starting from the second period of 32cc, MUX6 selects the output of $Q_i\_R_{ii}$ *Component* and MUX7 selects $AugH_4$ at the second clock cycle of every 4cc. The time sequence is shown in blue arrows at the lower left corner of Fig. 4.13. $R_{ij}\_H_jUpd$ *C*2 works in the same way as $R_{ij}\_H_jUpd$ *C3* during the first 64cc. After that, $AugH_4\_Upd\_2^{nd}$ is computed based on MUX4 and MUX5 at the third clock cycle of every 4cc. Similarly, since 96cc, $R_{ij}\_H_jUpd$ *C*1 works for three clock cycles on every 4cc. The scheduling of select signals for MUX2~ 7 is listed in Table 4.8.

Third, in the upper right corner of Fig. 4.13, there is a hardware unit to calculate G_{MMSE}. Several PDEs are inserted after the DEMUX, because Q_i (i=1, 2, 3, 4) are sent out at different time instances. When to calculate G_{MMSE} is controlled by Calc_ctr signal which is synchronous with Q_4. It is effective for 1cc on every 4cc from state S5 in Table 4.8 (b).

Table 4.8(a) Scheduling of select signals for MUX 1 ~ 3

name: output (control signal) / input No.: meaning		MUX1: Hi (QiRiiC_Ctr)	MUX2: Qi (Qi_ctr_1)	MUX3: Hj (HjUp_ctr_1)
S1: 0~31cc (i=0~7)	4i	**IN2:** *AugH1*	**IN2:** "1…1"	**IN1:** "1…1"
	4i+1	**IN1:** "1…1"		
	4i+2			
	4i+3			
S2: 32~63cc (i=8~15)	4i	**IN2:** *AugH1*	**IN2:** "1…1"	**IN1:** "1…1"
	4i+1	**IN3:** $AugH_2\_Upd\_1^{st}$	**IN1:** Q1	**IN2:** *Aug H2*
	4i+2	**IN1:** "1…1"	**IN2:** "1…1"	**IN1:** "1…1"
	4i+3	**IN1:** "1…1"	**IN2:** "1…1"	**IN1:** "1…1"

S3: 64~95cc (i=16~23)	4i	**IN2:** *AugH1*	**IN2:** "1…1"	**IN1:** "1…1"
	4i+1	**IN3:** *AugH$_2$_Upd_1st*	**IN1:** Q1	**IN2:** *Aug H2*
	4i+2	**IN3:** *AugH$_3$_Upd_2nd*	**IN1:** Q2	**IN3:** *AugH$_3$_Upd_2nd*
	4i+3	**IN1:** "1…1"	**IN2:** "1…1"	**IN1:** "1…1"
S4: 96~127cc (i=24~31)	4i	**IN2:** *AugH1*	**IN2:** "1…1"	**IN1:** "1…1"
	4i+1	**IN3:** *AugH$_2$_Upd_1st*	**IN1:** Q1	**IN2:** *Aug H2*
	4i+2	**IN3:** *AugH$_3$_Upd_2nd*	**IN1:** Q2	**IN3:** *AugH$_3$_Upd_2nd*
	4i+3	**IN3:** *AugH$_4$_Upd_3rd*	**IN1:** Q 3	**IN3:** *AugH$_4$_Upd_3rd*
S5: every 32cc after 127cc		same as S4		

Table 4.8 (b) Scheduling of select signals for MUX 4~7 and G_calc

name: output (control signal) / input/output / FSM state		RijHjUp C2		RijHjUp C3		G_calc *: (Calc_ctr)
		MUX4: *Qi* (Qi_ctr_2)	MUX5: *Hj* (HjUp_ctr_2)	MUX6: *Qi* (Qi_ctr_3)	MUX7: *Hj* (HjUp_ctr_3)	
S1: 0~31cc (i=0~7)	4i	IN2: "1…1"	IN1: "1…1"	IN2: "1…1"	IN1: "1…1"	no
	4i+1					
	4i+2					
	4i+3					
S2: 32~63cc (i=8~15)	4i	IN2: "1…1"	IN1: "1…1"	IN2: "1…1"	IN1: "1…1"	no
	4i+1	IN1: *Q1*	IN2: *AugH3*	IN1: Q1	IN2: *AugH4*	
	4i+2	IN2: "1…1"	IN1: "1…1"	IN2: "1…1"	IN1: "1…1"	
	4i+3					
S3: 64~95cc (i=16~23)	4i	IN2: "1…1"	IN1: "1…1"	Same as S2	Same as S2	no
	4i+1	IN1: *Q1*	IN2: *AugH3*			
	4i+2	IN1: *Q2*	IN3: *AugH$_4$_Upd_2nd*			
	4i+3	IN2: "1…1"	IN1: "1…1"			
S4: 96~127cc (i=24~31)	4i	Same as S3	Same as S3	Same as S2	Same as S2	no
	4i+1					
	4i+2					
	4i+3					
S5: every 32cc after 127cc	4i	all same as S4				yes
	4i+1					no
	4i+2					no
	4i+3					no

* "no" means no G_{MMSE} calculation.

From the analysis above, we can see that every 32cc period could be considered as a processing state of FSM. The detailed description of the FSM is shown in Fig. 4.14.

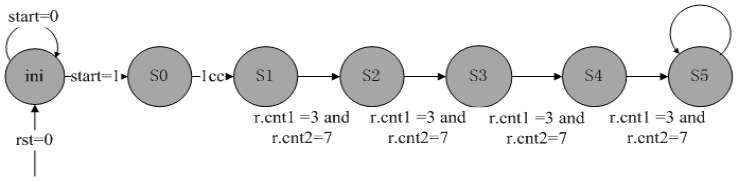

Fig. 4.14 FSM of the second pipeline architecture using new input scheduling

- ini: initial state when reset is low and wait for high start signal.
- S0: set control signal of MUX1 so that it will be effective in the next clock cycle. By doing this, $Q_i\_R_{ii}$ *Component* is able to receive the first $AugH_1$ vector at the beginning of S1 state.
- S1 (0~31cc as described above): this is the first period of 32cc in which $Q_i\_R_{ii}$ *Component* receives an $AugH_1$ every 4cc to calculate Q_1. Two counters are designed: cnt1 is from 0 to 3 and cnt2 is from 0 to 7, thus 4cc timing could be controlled by cnt1 and the period of 32cc can be obtained by combining cnt1 and cnt2.
- S2 (32~63cc): this is the second period of 33cc. $Q_i\_R_{ii}$ *Component* receives an $AugH_1$ or an $AugH_2\_Upd\_1^{st}$ every 4cc to calculate Q_1 and Q_2 respectively. Three $R_{ij}\_H_jUpd$ *Components* work for R_{1j} and $AugH_j\_Upd\_1^{st}$ (j=2, 3, 4).
- S3 (64~95cc): the third period of 32cc. $Q_i\_R_{ii}$ *Component* receives an $AugH_1$, an $AugH_2\_Upd\_1^{st}$ or $AugH_3\_Upd\_2^{nd}$ every 4cc to calculate Q_1 , Q_2 and Q_3. Besides R_{1j} and $AugH_j\_Upd\_1^{st}$ (j=2, 3, 4), three $R_{ij}\_H_jUpd$ *Components* also work for R_{2j}/ $AugH_j\_Upd\_2^{nd}$ (j=3 and 4).
- S4 (96~127cc): the fourth period of 32cc, $Q_i\_R_{ii}$ *Component* receives an $AugH_1$, an $AugH_2\_Upd\_1^{st}$, an $AugH_3\_Upd\_2^{nd}$, or $AugH_4\_Upd\_3^{rd}$ every 4cc to calculate Q_i (i=1, 2, 3, 4). $R_{ij}\_H_jUpd$ $C1$ is responsible for the calculation of R_{34}/ $AugH_4\_Upd\_3^{rd}$.
- S5 (every 32cc since 128cc): $Q_i\_R_{ii}$ *Component* and three $R_{ij}\_H_jUpd$ *Components* work in the same way as S4. The difference is that a G_{MMSE} matrix is calculated every 4cc in this state.

4.4.3 Implementation and Simulation

Same as the previous two schemes, a “diff” signal is used to indicate whether the result is correct or not. The behavioral simulation is given first in Fig. 4.15(a) for time sequence demonstration. It can be seen that the system begins to produce G_{MMSE} matrices on every 4cc since state S5.

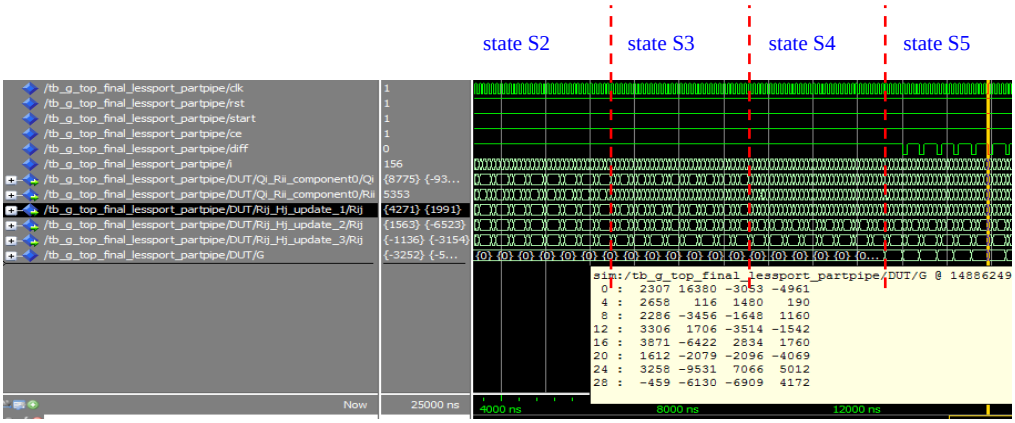

Fig. 4.15(a) Behaviral simultion of the second pipeline scheme

The minimum period is 95.941ns (Maximum Frequency: 10.423MHz) is reported from hardware synthesis. The post PNR simulation with this minimum period is shown in Fig. 4.15(b). The fact that “diff” turns to zero every 4cc demonstrates the expected results after PNR.

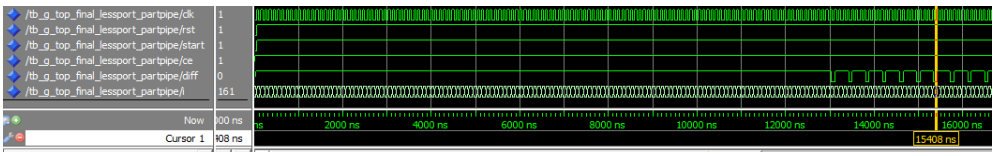

Fig. 4.15(b) Post PNR simultion of the second pipeline scheme

4.5 Comparisons of three architectures

In this section, three implementation schemes are compared.

First, Table 4.9 summarizes device utilizations of three architectures implemented using the same FPGA.

It can be seen that the time-multiplexed scheme requires the least hardware resources. The second pipeline architecture consumes about twice of the hardware resources of the time-multiplexed version. The first pipeline version is the most hardware demanding one, i.e., its device utilization is almost doubled in comparison to that of the second pipeline design.

Table 4.9 Device utilization comparison of the three schemes

slice logic utilization	the 1st pipeline	time-multiplexed	the 2nd pipeline
number of slice registers	15,680	3,363	6,901
number used as Flip Flops	14,536	2,799	6,105
number used as AND/OR logics	1,144	564	796
number of slice LUTs	17,105	4,504	9,958
number used as logic	13,730	4,409	8,010
number used as Memory	3,045	3	1,926
number used exclusively as route-thrus		92	
number of occupied slices	5,575	1,750	3,652
number of LUT flip flop pairs used	19,565	5,359	10,783
number with an unused Flip Flop	4,893	2,343	4,033
number with an unused LUT	2,460	855	825
number of fully used LUT-FF pairs	12,212	2,161	5,925
number of unique control sets	17	18	19
number of slice register sites lost to control set restrictions	75	38	57
DSP48E1s	636	240	368

Second, the throughputs are summarized in Table 4.10. It can be seen that the throughput using the first pipeline architecture is 3 times higher than the second pipeline architecture and 135 times higher than that of the time-multiplexed scheme.

Table 4.10 Throughput comparison of the three schemes

	the 1st pipeline	time-multiplexed	the 2nd pipeline
throughput	1 G_{MMSE} per cc	1 G_{MMSE} per 136cc	1 G_{MMSE} per 4cc

Finally, regarding maximum frequency, time-multiplexed and the second pipeline architecture are almost the same. See Table 4.11. The frequency limitation of the square-root module is the main bottleneck of all three schemes.

Table 4.11 Maximum frequency comparison of three schemes

	the 1st pipeline	time-multiplexed	the 2nd pipeline
maximum frequency	15.734MHz	10.413MHz	10.423MHz

Given the throughput and frequency, pipeline version 1 has the highest processing speed which is 5 times faster than the pipeline version 2 and more than 200 times faster than the time-multiplexed scheme. If a system does not require high throughput, the second pipeline architecture is a good choice for its reasonable performance and area consumption. Due to its low throughput in comparison to the other two architectures, time-multiplexed scheme should be considered only when device resource is limited.

CHAPTER 5

5 Conclusion and Future Work

5.1 Conclusion

Motivated by implementation of MIMO detection algorithms on RCA and ASIC platforms, this thesis mainly studies two linear MMSE MIMO detection algorithms: squared MMSE algorithm and QRD-based square-root MMSE algorithm. Besides, three matrix inversion approaches required by squared MMSE are studied, including DMI, analytic/block-wise, and QRD-based inversion.

First, considering computational complexity, regardless of the antenna size, squared MMSE using analytic or block-wise inversion has the least operation numbers when only considering matrix inversion. When including steps required by calculating both R and G_{MMSE}, square-root MMSE scheme with QRD is the least compute-intensive one. In terms of architecture, implementation on ASICs generally requires fewer operations than that on RCAs. However, their difference reduces as antenna number increases.

Second, accuracy is investigated in both floating-point and fixed-point formats. Comparisons show that QRD-based square-root MMSE scheme with dynamic scaling provides the highest accuracy among four schemes with respect of floating-point. Moreover, it shows a better performance than QRD-based squared MMSE considering fixed-point format.

Finally, to implement QRD-based square-root MMSE that has highest accuracy and lowest complexity, three architectures are designed and evaluated, including two pipeline versions and a time-multiplexed version. The first pipeline version is able to produce one result every clock cycle. However, it consumes the most hardware resources among the three schemes. The time-multiplexed version is the least expensive one in terms of hardware size, but its throughput is less than 1 percent of the first pipeline version due to data dependency of the MGS algorithm and 33cc latency of the divider. The second pipeline architecture shows a good performance trade-off among the three. Its processing throughput is 33 times higher than that of the time-multiplexed version, and its hardware consumption is about half of that of the first pipeline version.

5.2 Future work

This thesis compares the computational complexity of different schemes implemented on ASIC and RCA platforms respectively. Hardware

implementations need to be further studied, so that a comprehensive comparison can be made between two platforms.

At the implementation level, the performance of square-root module can be improved. This function unit can be re-designed by adopting algorithms such as Newton-Raphson method.

References

[1] N. Morinaga, R. Kohno, S. Sampei, *Wireless Communication Technologies: New Multimedia Systems*, Kluwer Academic Publishers, 0-7923-7900-4, 2000

[2] C. Zhang, L. Liu, and V. Öwall, "Mapping Channel Estimation and MIMO Detection in LTE-Advanced on a Reconfigurable Cell Array", in the *proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp 1799–1802, 978-1-4673-0218-0, May 2012

[3] L. Bai, and J. Choi, *Low Complexity MIMO Detection*, New York, Springer Science + Business Media, 978-1-4419-8583-5, 2012

[4] H. S. Kim, W. Zhu, J. Bhatia, K. Mohammed, A. Shah, and B. Daneshrad, "A Practical, Hardware Friendly MMSE Detector for MIMO-OFDM-Based Systems", *EURASIP Journal on Advances in Signal Processing*, vol. 2008, no. 94, pp 1-14, 1867-6180, March 2008

[5] C. Zhang, T. Lenart, H. Svensson and V.Öwall, "Dynamically Reconfigurable Architectures for Real-time Baseband Processing", in the *proceedings of 2009 International Conference on Reconfigurable Computing and FPGAs*, pp338-343, 978-1-4244-5293-4 , December 2009

[6] K. Kuttler, *Linear Algebra: Theory and Applications*, The Saylor Foundation: Wave I of the Open Textbook Challenge, CC-BY3.0, 2012

[7] D. Wu, J. Eilert, D. Liu, D. Wang, N. Al-Dhahir, and H. Minn, "Fast Complex Valued Matrix Inversion for Multi-User STBC-MIMO Decoding", in the *proceedings of IEEE Computer Society Annual Symposium on VLSI*, ISVLSI'07, pp 325–330, 0-7695-2896-1, March 2007

[8] L. Fahrmeir, T. Kneib, S. Lang, B. Marx, *Regression: Models, Methods and Applications*, Springer Heidelberg, New York, 978-3-642-34333-9, 2013

[9] N. J. Higham, *Accuracy and Stability of Numerical Algorithms: Second Edition*, Philadelphia, the Society for Industrial and Applied Mathematics, 0-89871-521-0, 2002

[10] W. Cheney, and D. Kincaid, *Linear Algebra: Theory and Applications*, Jones and Bartlett Learning, Mississauga, 978-1-4496-1352-5, 2012

[11] A. El-Amawy and K.R. Dharmarajan, “Parallel VLSI algorithm for stable inversion of dense matrices” in the *proceedings of IEE Computers and Digital Techniques*, Vol. 136, No. 6, pp 575–580, 0143-7062, NOVEMBER 1989

[12] A. Burg, S. Haene, D. Perels, P. Luethi, N. Felber and W. Fichtner, “Algorithm and VLSI Architecture for Linear MMSE Detection in MIMO-OFDM Systems”, in the *proceedings of 2006 IEEE International Symposium on Circuits and Systems*, 0-7803-9389-9, pp 4102-4106, Island of Kos, Greece, May 2006

[13] J. Moon, H. Jin, T. Jeon, and S. Lee, “Channel estimation for MIMO-OFDM systems employing spatial multiplexing”, in the *proceedings of IEEE Vehicular Technology Conference*, Vol. 5, pp 3649-3654, 0-7803-8521-7, September 2004

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2015-426

http://www.eit.lth.se