UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GABRIEL LUCA NAZAR

QR Decomposition Algorithms for MIMO Systems: Impact on Computational Effort and Hardware Implementations

Trabalho de Diplomação.

Prof. Dr. Luigi Carro
Orientador

Porto Alegre, abril de 2009.

ACKNOWLEDGEMENTS

I would like to thank Christina Gimmler and Prof. Dr.-Ing. Norbert Wehn, from TU Kaiserslautern, for the support and guidance during my stay in Germany. I would also like to thank Prof. Dr. Luigi Carro for being my advisor on this work after my return to Brazil.

And last but not least, I would like to thank my family and friends for the support. And one very special thank to Bianca for always believing in me.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application Specific Integrated Circuit
BCJR	Bahl, Cocke, Jelinek and Raviv
CS	Constant Sphere
ES	Early Subtraction
FER	Frame Error Rate
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
IS	Intermediate Subtraction
LDPC	Low-Density Parity Check
LLR	Logarithmic Likelihood Ratio
LS	Late Subtraction
LUT	Look-Up Table
MAP	Maximum A-Posteriori
MIMO	Multiple Input Multiple Output
MMSE	Minimum Mean-Square Error
NR	Newton-Raphson
NU	Norm Update
OSS	Ordered Sphere Shrinking
PED	Partial Euclidean Distance
QAM	Quadrature Amplitude Modulation
SISO	Soft Input Soft Output
SNR	Signal-to-Noise Ratio
SQRD	Sorted QR Decomposition
SS	Sphere Shrinking
VHDL	Very-high-speed integrated circuit Hardware Description Language

LIST OF FIGURES

LIST OF TABLES

Algoritmos de decomposição QR para sistemas MIMO: Impacto no esforço computacional e implementações de hardware

RESUMO

Dentre as abordagens para se atingir altas taxas de transmissão em sistemas de comunicação sem fio, uma se destaca como muito promissora: sistemas de múltiplas antenas (ou Multiple Input Multiple Output – MIMO), nos quais a informação é transmitida e recebida por mais de uma antena. Tais sistemas podem atingir altas taxas de transmissão usando, entre outras possibilidades, algoritmos de *sphere decoding* para decodificar os símbolos MIMO recebidos.

Para diversos algoritmos para detecção MIMO, tal como sphere decoding, uma variação do algoritmo de Fincke-Pohst (FINCKE, 1985), é necessário ter um hardware eficiente de decomposição QR, uma vez que esse é utilizado cada vez que a resposta impulsiva do canal modifica-se significativamente. E para obter-se uma implementação eficiente é necessário utilizar uma representação com ponto fixo para as matrizes, tanto por motivos de área quanto de latência.

Evidentemente, a perda de precisão resultante do uso de ponto fixo introduz erros nas matrizes calculadas, e é provável que isso leve a um aumento na taxa de erros de quadros (FER). Um dos propósitos deste trabalho é determinar a quantidade mínima de bits necessária para manter esse aumento suficientemente baixo. Este trabalho também avalia a redução no esforço computacional para execução de detecção MIMO por algoritmos baseados em busca em árvore resultante do uso de versões melhoradas do algoritmo de decomposição QR. Mais especificamente, a sorted QR decomposition (SQRD) e a minimum mean-square error SQRD são avaliadas.

O outro propósito deste trabalho é projetar arquiteturas de hardware capazes de computar a decomposição QR e suas variações para matrizes pequenas, tipicamente de 2ª e 4ª ordem. Também é importante obter uma descrição em VHDL desse hardware e comparar resultados de área e latência das diferentes versões.

Palavras-Chave: Decomposição QR, SQRD, MMSE-SQRD, Sphere Decoding.

ABSTRACT

Among the approaches to achieve high data rates in wireless systems, one rises as very promising: multiple-antenna systems (or Multiple Input Multiple Output – MIMO), in which the information is transmitted and received by multiple antennas. Such systems can achieve high data rates with using, among other possible choices, sphere decoding algorithms to decode the received MIMO symbols.

For many algorithms used for MIMO detection, such as sphere decoding, a variation of the Fincke-Pohst Algorithm (FINCKE, 1985), it is required to have an efficient QR decomposition hardware, since it is used each time the channel impulse response changes significantly. And to achieve an efficient hardware implementation it is necessary to use a fixed point representation for the matrices, both for area and latency purposes.

Evidently, the loss in precision resultant from fixed point precision introduces errors in the output matrices, and this is likely to lead to an increase in the frame error rate (FER). One of the purposes of this work is to determine the minimum amount of bits both for fractional and integer parts that are necessary to keep this increase sufficiently low. This work also evaluates the complexity reduction resultant from improved versions of the QR decomposition in tree-based search algorithms. More specifically, the sorted QR decomposition (SQRD) and the minimum mean-square error SQRD are evaluated.

The other main purpose of this work is to come up with hardware architectures capable of computing the QR decomposition and its improved versions (SQRD and MMSE-SQRD) for small matrices, typically of 2^{nd} and 4^{th} order. It is also important to have a fully functional VHDL description of this hardware and compare the different versions regarding area and latency.

Keywords: QR decomposition, SQRD, MMSE-SQRD, Sphere Decoding.

1 INTRODUCTION

1.1 Context and Motivation

Devices such as smart phones, laptops and other similar mobile communication gadgets are becoming growingly common. The applications heterogeneity is also increasing, since such devices can be used from applications as simple as sending text messages or reading e-mail up until watching live video streams. Many of these applications, especially those involving multimedia streams, require high data rates to be performed with satisfactory quality. Therefore, there is an increasing demand for devices able to transmit and receive efficiently at high speed and better techniques are required to increase spectral efficiency, while reducing error rates and decoding complexity.

In order to reach the expected data rates for future wireless systems, multiple-input multiple-output (MIMO) systems rise as one of the most promising techniques (GIMMLER, 2007), (LUETHI, 2008). In MIMO systems, multiple transmit antennas send MIMO symbols over the channel in the same frequency band, which are received also by multiple antennas. Such systems are especially attractive due to their high spectral efficiency (TELATAR, 1999). On the other hand, these systems have the drawback of increased complexity in the receiver, since it is responsible for determining which modulation symbol was sent by each transmit antenna.

Many different algorithms can be used to reduce the complexity of detecting MIMO symbols, such as sphere decoding and successive interference cancellation. For many of these algorithms, the QR decomposition is a critical operation to ensure a good quality of the successive decoding steps. Also, improved versions of the QR decomposition can be used to further improve the detection quality, reducing frame error rates or computational effort, depending on the detection algorithm used (WÜBBEN, 2001), (MENNENGA, 2009). A number of different QR decomposition architectures were proposed in other works, such as (SALMELA, 2008) and (LUETHI, 2008). In these works, however, the results are presented regarding the architecture itself, not addressing the overall system performance.

1.2 Contributions

The extended versions of the QR decomposition evaluated, namely the sorted QR decomposition (SQRD) and the minimum mean-square error SQRD (MMSE-SQRD) can reduce the total computational effort associated with detecting a MIMO symbol, when tree-search based algorithms are used (MENNENGA, 2009). However, the actual reduction that can be achieved must still be measured and compared for each different version, concerning the complete communication chain. Also, the cost of using these algorithms must be evaluated, regarding increase in area and decomposition latency.

Therefore, in this work the computational effort and error rates of different versions of the QR decomposition are measured using a communication chain model. In order to obtain a fixed point implementation that is sufficiently precise, the error rates of different quantizations is also analysed.

A new QR decomposition hardware architecture is presented. The proposed architecture is extended to perform the SQRD and MMSE-SQRD. The implementation results are presented, regarding total area and latency, which is compared to the requirements of a real system's real time requirements. The results are also compared to those obtained from a high-level synthesis tool, the Mentor Graphics Catapult C.

1.3 Report Structure

The remainder of this report is structured as follows. Chapter 2 presents the communication chain model and briefly describes each of its components. Chapter 3 describes the chosen QR decomposition algorithm and the investigated improvements. Chapter 4 explains the different simulation parameters and presents several simulation results. In chapter 5 the QR decomposition hardware architectures are presented and each of the required components is described. Chapter 6 analyses the area and timing results of the hardware implementations for FPGAs and ASICs. Finally, chapter 7 presents the conclusions.

2 COMMUNICATION CHAIN MODEL

In order to evaluate the functionality and the results obtained from modifications applied to the system, a communication chain model is required. The used model consists of: source, channel encoder, interleaver, QAM mapper, channel, sphere decoder, de-interleaver and channel decoder.

Figure 2.1 shows the basic chain. The source generates a random bit sequence **b**, in which each bit has an equal probability of being 1 or 0. This generated sequence is the input of the channel encoder, which can use many different algorithms, such as LDPC or convolutional codes. Since the evaluation of such algorithms is not in the scope of this work, only the latter is used. The used convolutional code has 64 states, and it is non-systematic, non-recursive and uses the Maxlog MAP (BCJR) algorithm. The encoded sequence **c'** is then interleaved. The interleaved sequence **c** is then mapped into a complex QAM symbols vector **s** by the QAM mapper, and then transmitted by N_T antennas over a noisy channel with Rayleigh fading, to be received by N_R antennas.

The received sequence **y** is then decoded by the sphere decoder, de-interleaved and then decoded by the channel decoder. This process is repeated iteratively, in the called big loop iterations. During this process, the sphere and channel decoders exchange logarithmic likelihood ratios (LLRs), which express the belief of each decoder for each bit being 1 or 0. The extrinsic information vector $\mathbf{e_{SD}}$, generated by the sphere decoder, is de-interleaved and then decoded by the channel decoder, which has two outputs: the $\hat{\mathbf{b}}$ bit vector, and the extrinsic information vector $\mathbf{e'_{CD}}$. The latter is interleaved and used by the sphere decoder in the next big loop iteration. Both decoders are called soft-input soft-output (SISO) decoders.

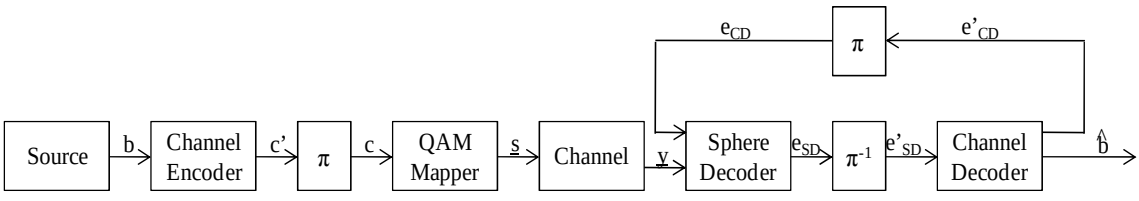

Figure 2.1: System model

2.1 Channel model

The channel model describes how the transmitted symbols are modified before the reception. Typically it involves the addition of noise and, for the multiple antennas case, the multiplication by a channel impulse response matrix **H**.

When compared to the one used in (GIMMLER, 2007), one important change was made in the channel model. Instead of

$$y = \sqrt{\frac{SNR}{N_T}} Hs + n \tag{2.1}$$

was used the model

$$y = \sqrt{\frac{1}{N_T}} Hs + \frac{n}{\sqrt{SNR}} \tag{2.2}$$

y : $N_R \times 1$, received symbol vector

H : $N_R \times N_T$, complex channel matrix

s : $N_T \times 1$, transmitted symbol vector

n : $N_R \times 1$, additive gaussian complex noise

This means that, instead of scaling the **H** matrix with the signal to noise ratio, the noise vector is divided by this same ratio. With this change we get a more predictable range for the values in **H**, since it is no longer scaled by the SNR. The other main difference caused by this change is the alteration of the sphere radius (SIMHA S, 2009), being much smaller than in (GIMMLER, 2007).

The **H** matrix and the noise vector **n** are both constituted of complex random values, with mean zero and unit variance. In the rest of this work it is assumed that $N_R = N_T = N$.

2.2 The Sphere Decoding algorithm

The sphere decoder has the purpose of determining the logarithmic likelihood ratio L_D of each bit b_j in the received complex vector **y**. This numbers form a vector which represents the belief of the sphere decoder for each bit being 1 or 0.

$$L_D(b_j) = \ln \frac{P(b_j = 0)}{P(b_j = 1)} \tag{2.3}$$

This means that, if $L_D(b_j) > 0$, the hard-decision is $b_j = 0$ and it is $b_j = 1$ if $L_D(b_j) < 0$. Accordingly, if $L_D(b_j) = 0$, no information can be extracted, as $P(b_j = 0) = P(b_j = 1)$.

To determine the probability of each bit being 1 or 0, a search has to be performed through the solution space, where the distances between the received vector and the possible solutions in the discrete QAM constellation are considered. Assuming that the receiver knows the channel matrix **H**, the squared Euclidean distance between the received **y** vector and each possible solution vector **s** is:

$$d(s) = \|y - Hs\|^2 \tag{2.4}$$

This distance is then updated with the logarithmic probabilities for each bit calculated in the previous big loop iteration by the channel decoder, and the sphere decoder then tries to find the symbol vector which results in the minimum distance:

$$\min(d(s) - \sum_j \log P(b_j)) \tag{2.5}$$

Where $P(b_j)$ denotes the probability of the j^{th} bit having the value that was assumed for it in **s**. In other words, the purpose of this operation is to reduce the distance of the symbols that the channel decoder agrees with. However, these logarithmic probabilities are not used in this work. Instead, logarithmic likelihood ratios (LLRs) are used to approximate this value. The search performed by the sphere decoder then becomes:

$$\min(d(s) + \sum_j LLR(a_j, b_j)) \tag{2.6}$$

where a_j is the a-priori information received from the channel decoder. The straightforward approach to find these minimum values is to try all possible values for **s**. However, it is clear that the computational effort required to analyse each possible **s** vector makes this approach very ineffective. For example, in a system with 4×4 antennas and a 16-QAM constellation, we would have 16^4 = 65536 possibilities. A variation of the Fincke-Pohst algorithm can be used to reduce this problem, analyzing through a tree search only the elements that reside inside a sphere of a given radius. This reduces the search space and makes it possible to recursively calculate the squared Euclidean distance using partial Euclidean distances (PEDs). In order to do so, the equation for the distance needs to be modified, using the QR decomposition of matrix **H**. This will be further discussed in section 2.2.1.

The output of the sphere decoder is also logarithmic likelihood ratios. They are calculated, however, as the difference of the minimum distance for this bit being 1 and 0:

$$\Lambda_j = \min 1_j - \min 0_j \tag{2.7}$$

These values keep the properties described for the L_D values in equation 2.3.

Because during the big loop iterations only the new information discovered by the sphere decoder should be sent forward to the channel decoder, the added a-priori information a_j of bit b_j must be subtracted from the difference of the minima vectors, to form the output $\mathbf{e_{SD}}$.

$$e_{SD\,j} = \Lambda_j - a_j \tag{2.8}$$

Since Λ_j is calculated as the mentioned difference, the a-priori information must be added in a manner that maintains the property in equation 2.9, so that its subtraction effectively removes its value from the result:

$$LLR(a_j, b_j = 1) - LLR(a_j, b_j = 0) = a_j \tag{2.9}$$

Therefore, the purpose of the LLR function in equation 2.6 is to transform the a_j values received in a vector which respects the property in 2.9. Several different implementations for this function are discussed in (GIMMLER, 2007). In this work, the version used adds only the a-priori information of bits that are 1:

$$LLR(a_j, b_j) = \begin{cases} a_j, b_j = 1 \\ 0, b_j = 0 \end{cases} \tag{2.10}$$

2.2.1 The usage of the QR decomposition

In order to simplify the calculation of the distance between the received vector and other constellation point vectors, the QR decomposition can be employed:

$$d(s) = \left\| y - Hs \right\|^2 \tag{2.11}$$

$$d(s) = \left\| y - QRs \right\|^2 \tag{2.12}$$

$$d(s) = \left\| Q^H y - Rs \right\|^2 \tag{2.13}$$

The QR decomposition of a matrix **H** is a mathematical operation that generates two matrices, namely **Q** and **R**, which have the following properties:

i) The columns of **Q** are orthogonal (or *orthonormal*). That means the inner product of any two columns of **Q** is zero, and the inner product of a column with itself is one (for the orthonormal case). From this fact derives the useful property that the inverse of **Q** is its transpose (or Hermitian transpose, if **Q** is complex).

ii) **R** is an upper-triangular matrix. That means that any element under the main diagonal is zero.

iii) Finally, the product of **Q** and **R** must be **H**.

$$H = QR \tag{2.14}$$

Using these definitions, a tree search can then occur in a tree with N+1 levels, with each level after the first standing for one element of the solution vector **s** and with each node having M children, where M is the number of possible constellation points in the used QAM modulation, as in Figure 2.2.

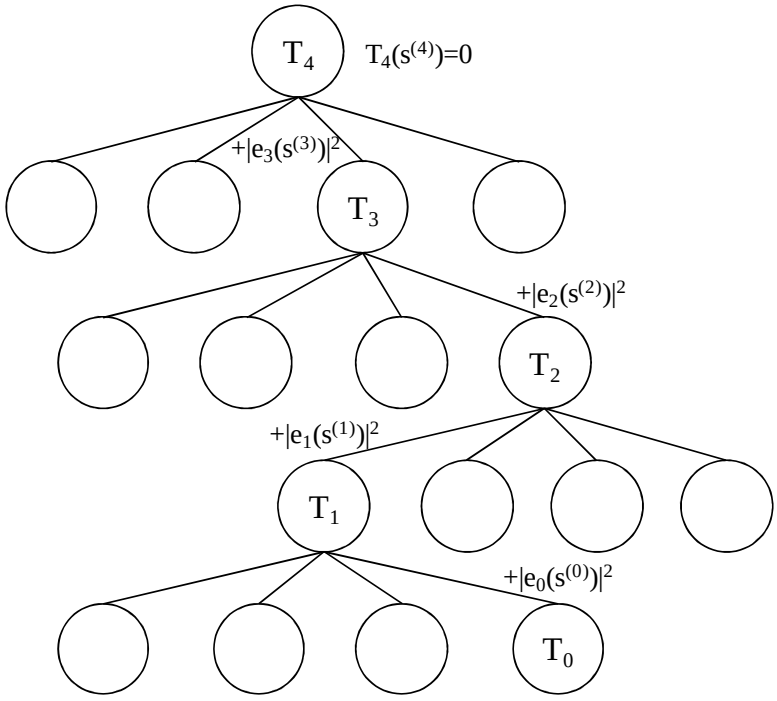

Figure 2.2: Tree search for a system with 4-QAM and 4×4 antennas

Starting at the root with a PED $T_N(s^{(N)}) = 0$, it is possible do build the tree downwards calculating the distances $T_i(s^{(i)})$ recursively. Once a leaf node is reached, its distance $T_0(s^{(0)})$ is actually the distance for the whole symbol sequence that connects it to root node. The a-priori information from the previous big loop iteration can also be added during process.

The partial Euclidean distances $T_i(s^{(i)})$ are calculated recursively with the equation:

$$T_i(s^{(i)}) = T_{i+1}(s^{(i+1)}) + \left|e_i(s^{(i)})\right|^2 \tag{2.15}$$

The partial increases $|e_i(s^{(i)})|^2$, already adding the a-priori information for the relevant bits, are obtained with:

$$\left|e_i(s^{(i)})\right|^2 = \left|\hat{y}_i - \sum_{j=i}^{N-1} R_{ij} s_j\right|^2 + \sum_k LLR(a_k, b_k) \tag{2.16}$$

$$\hat{y} = Q^H y \tag{2.17}$$

Where k iterates the bits which form the current symbol s_i. To emphasize the effect of s_i in the partial Euclidean distance increase, equation 2.16 can be rewritten as:

$$\left|e_i(s^{(i)})\right|^2 = \left|b_{i+1}\left(s^{(i+1)}\right) - R_{i,i} s_i\right|^2 + \sum_k LLR(a_k, b_k) \tag{2.18}$$

$$b_{i+1}\left(s^{(i+1)}\right) = \hat{y}_i - \sum_{j=i+1}^{N-1} R_{i,j} s_j \tag{2.19}$$

Note that equation 2.19 is equal for all brother nodes.

The sphere radius can be applied during this search: an intermediary node with a high PED can have its whole sub-tree excluded from the search, reducing the amount of evaluated possibilities and saving computational time.

2.2.2 Sphere Shrinking

One important improvement that can be applied to the sphere decoding algorithm is sphere shrinking. The main purpose of this technique is to reduce the amount of visited nodes by reducing the sphere radius as the tree search proceeds.

At any given stage of the tree search, it is possible to determine the maximum value of the distance of a leaf node that may cause an alteration in the minima vectors. The sphere shrinking radius SSR is equal to the largest value currently in a minima vector, since that any distance larger than this will not modify any position in the vectors and is, therefore, irrelevant for the output. Thus, if a node reaches this value at any point of the search, one can know that the entire sub-tree below it is irrelevant – which is the same as redefining the sphere radius. Evidently, if the initial radius ISR is smaller than this new value, it is kept.

$$SR = \min(ISR, SSR) \tag{2.20}$$

$$SSR = \max(\max(\min_0), \max(\min_1)) \tag{2.21}$$

At this point, however, it is important to consider the manner in which the a-priori information is added, as it can be negative. This means that an intermediate node which lies outside the sphere can have a final point (or leaf node) under it which would cause an update in a minima vector, since negative a-priori information can cause the distance of a given branch in the tree to actually decrease as the search proceeds, provided it overcomes the positive PED. Even though this may occur with considerable frequency, its influence in the final error rates is not significant.

Among the LLR functions presented in (GIMMLER, 2007), one ensures that the a-priori information is always positive (equation 2.22), which does not happen when

equation 2.10 is used. However, equation 2.10 has other advantages: the amount of additions done is only the half of equation 2.22 and it needs to visit fewer nodes to achieve the same FER performance (GIMMLER, 2007).

$$LLR(a_j, b_j) = \begin{cases} a_j, b_j = 1 \wedge a_j \geq 0 \\ -a_j, b_j = 0 \wedge a_j < 0 \\ 0, otherwise \end{cases} \quad (2.22)$$

Other improvements can be applied to the sphere shrinking. As the search proceeds down the tree, decisions are made for the value of each bit. This means that leaf nodes reached in the current sub-tree cannot affect the minima of the other value for the already decided bits. Hence, the definition of the SSR in equation 2.21 can be further improved to consider only the minima of the chosen values for the bits that lie upwards in the tree. Bits that are still to have their values chosen down the tree must have both minima considered. Figure 2.3 illustrates this for a system with 4×4 antennas and 4-QAM. Consider that the tree search is currently in the gray node, labelled T_1, and needs to evaluate whether it is inside the sphere or not. The first six bits of the sequence already have their values defined, and therefore only those values can have their minima changed. The last two, on the other hand, are still undefined. For this reason, only the minima in gray need to be considered to determine the current radius, as they are the only ones that are eligible to be changed. The maximum value in the gray painted positions will be used as the sphere radius.

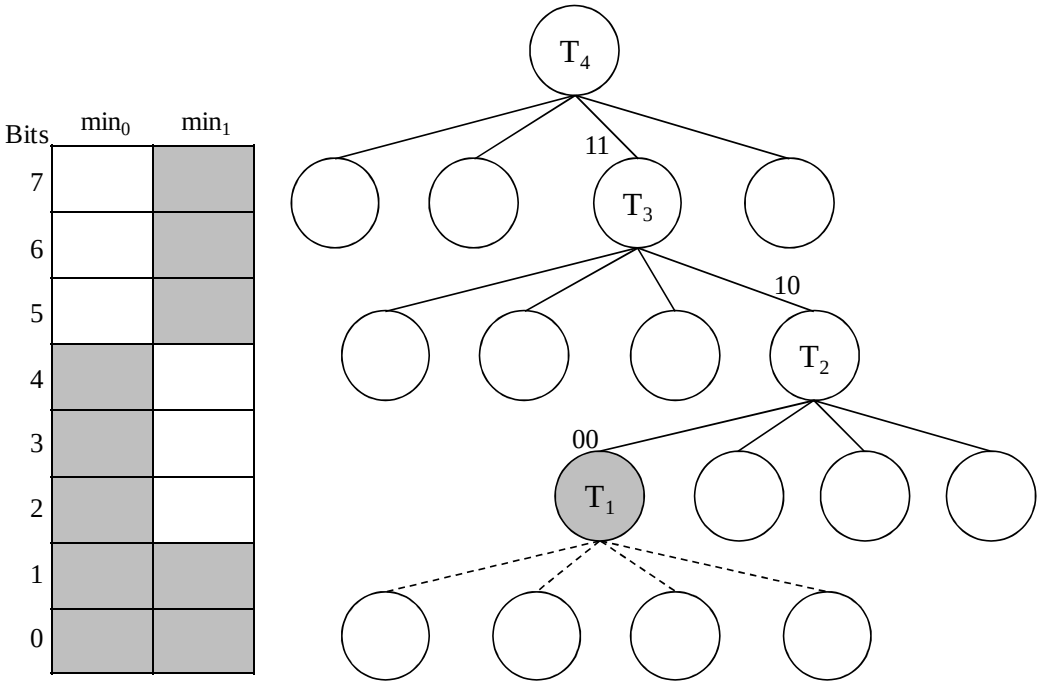

Figure 2.3: Sphere shrinking example for a 4×4 antennas system with 4-QAM

The last improvement of the sphere shrinking to be considered in this work is the ordering of the child nodes. This is called *ordered sphere shrinking*. The idea is to always choose the node with the smallest PED as the first candidate when descending the tree. This is a greedy algorithm used to obtain quickly leaf nodes with reasonably small distances and hence speed-up the shrinkage of the tree, further reducing the average amount of visited nodes.

3 QR DECOMPOSITION ALGORITHM

The QR decomposition of a matrix **H** is defined as the matrices **Q** and **R**, where **Q** has orthonormal columns, **R** is upper-triangular and **H**=**QR**. There are many different algorithms to compute the QR decomposition, such as using the Gram-Schmidt process, Householder reflections or Givens rotations and variations (GOLLUB, 1996). In this work, the Gram-Schmidt process is used, due to its simplicity and vast use in previous works, such as (SALMELA, 2008), (WÜBBEN, 2001) and (LUETHI, 2008).

3.1 The Gram-Schmidt process

The Gram-Schmidt process initially orthogonalizes a set of vectors by subtracting the projection of one on the other. Figure 3.1 shows the process for two vectors.

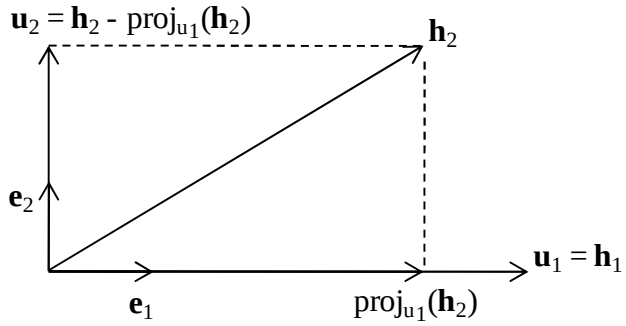

Figure 3.1: Gram-Schmidt process for two vectors

To perform the process, the definitions of inner product and vector projection are necessary. Since we are working in an Euclidean space, the inner product of two vectors **h** and **u** is defined as the dot product:

$$\langle h,u\rangle = h[1].u[1] + h[2].u[2] + ... + h[N].u[N] \tag{3.1}$$

If the vectors are complex, which is the case here, a slight alteration in this equation is necessary, where *conj* denotes the complex conjugate of a number:

$$\langle h,u\rangle = conj(h[1]).u[1] + conj(h[2]).u[2] + ... + conj(h[N]).u[N] \tag{3.2}$$

The projection of a vector **h** on a vector **u** can be defined as:

$$proj_u(h) = \frac{\langle u,h\rangle}{\langle u,u\rangle}u \tag{3.3}$$

Given these definitions, the Gram-Schmidt process is computed as follows, iteratively subtracting the projection of each vector on the others:

$$
\begin{aligned}
u_1 &= h_1 \\
u_2 &= h_2 - proj_{u1}(h_2) \\
u_3 &= h_3 - proj_{u1}(h_3) - proj_{u2}(h_3) \\
&\vdots \\
u_k &= h_k - \sum_{j=1}^{k-1} proj_{uj}(h_k)
\end{aligned} \tag{3.4}
$$

We can consider that the vectors $\mathbf{h}_k$ are the columns of the input matrix $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N)$, which were transformed into a set of orthogonal vectors $\mathbf{u}_k$. This set of vectors can be then transformed into an orthonormal set by dividing each element by its own modulus:

$$e_k = \frac{u_k}{\|u_k\|} \tag{3.5}$$

Since the inner product of two vectors is directly proportional to the norm of each vector, and the inner product of a vector with itself is its squared norm, the projections in equation 3.4 can be rewritten as:

$$proj_{u_j}(h_k) = \frac{\langle u_j, h_k \rangle}{\|u_j\|^2} u_j = \frac{\langle u_j, h_k \rangle}{\|u_j\|} e_j = \langle e_j, h_k \rangle e_j \tag{3.6}$$

Equation 3.4 can be rewritten using equation 3.6, the fact that $\langle e_i, h_i \rangle e_i = u_i$, since e_i has norm 1 and is collinear to u_i, and isolating the elements $\mathbf{h}_k$:

$$
\begin{aligned}
h_1 &= \langle e_1, h_1 \rangle e_1 \\
h_2 &= \langle e_1, h_2 \rangle e_1 + \langle e_2, h_2 \rangle e_2 \\
h_3 &= \langle e_1, h_3 \rangle e_1 + \langle e_2, h_3 \rangle e_2 + \langle e_3, h_3 \rangle e_3 \\
&\vdots \\
h_k &= \sum_{j=1}^{k} \langle e_j, h_k \rangle e_j
\end{aligned} \tag{3.7}
$$

Rewriting the above equations in matrices results in:

$$
H = (e_1 \mid e_2 \mid \ldots \mid e_N) \begin{pmatrix} \langle e_1, h_1 \rangle & \langle e_1, h_2 \rangle & \ldots & \langle e_1, h_N \rangle \\ 0 & \langle e_2, h_2 \rangle & \ldots & \langle e_2, h_N \rangle \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & 0 & \langle e_N, h_N \rangle \end{pmatrix} \tag{3.8}
$$

As the vectors $\mathbf{e}_k$ are orthonormal and the second matrix in the above product is upper triangular, it can be seen that the first and second matrices in the right hand side of equation 3.8 are, respectively, **Q** and **R**.

3.2 Modified Gram-Schmidt process

The direct application of the Gram-Schmidt process in a system with finite precision (such as any computational system) yields poor results. More specifically, the

orthogonality of the columns of **Q** is seriously damaged as the precision is reduced (GOLLUB, 1996).

These results can be improved by an alteration in the ordering in which the calculations are done, resulting in the algorithm called Modified Gram-Schmidt (MGS) (GOLLUB, 1996), which calculates, at each iteration of its external loop, one column of the **Q** matrix and one row of the **R** matrix.

Let **H** denote the input matrix, $\mathbf{h}_k$ denote the k^{th} column of **H** and $\mathbf{q}_k{}^H$ denote the Hermitian transpose of $\mathbf{q}_k$. A pseudo-code for this algorithm is:

```
R = 0
for k=0:N-1
        R(k, k) = || h_k ||
        q_k = h_k / R(k, k)
        for j=k+1:N-1
                R(k, j) = q_k^H . h_j
                h_j = h_j – q_k .R(k, j)
        end
end
```

Algorithm 1: Modified Gram-Schmidt Process (MGS)

This version of the algorithm, however, has some visible hardware implementation issues. First, it requires the calculation of the norm of a vector, which implies the need of a square root hardware. Second, there is the need for a division hardware. And, if this vector division operation is ever to be parallelized, then many instances of this hardware would be required.

One approach to avoid these implementation issues is presented at (SALMELA, 2008). Instead of calculating the norm of the vector, the inverse square root is calculated directly. This allows the replacement of the divisions by multiplications. The actual square root, necessary to form the main diagonal of the **R** matrix, can be calculated using the mathematical property:

$$\sqrt{x} = x^{1/2} = xx^{-1/2} = x\frac{1}{\sqrt{x}} \tag{3.9}$$

This means that multiplying the input and the output of the inverse square root hardware results in the square root of the input. The gain in area from this alteration can be even larger if an approximation hardware is used, rather than an exact one, to calculate the inverse square root. The hardware used for this purpose is further discussed and presented in chapter 5. However, in order to keep this approximation close enough to the correct result, numeric methods such as the Newton-Raphson method can be used.

Given a function f(y), its derivative f'(y) and an initial guess for a root y_0, the Newton-Raphson method can be used to improve this guess, iteratively finding a better root approximation (PRESS, 1992):

$$y_{i+1} = y_i - \frac{f(y_i)}{f'(y_i)} \tag{3.10}$$

The equation for the inverse square root must then be rewritten:

$$\frac{1}{\sqrt{x}} = y \tag{3.11}$$

$$\frac{1}{y^2} = x \tag{3.12}$$

Then the function f(y) must be defined in a way that when f(y) = 0, y = $1/\sqrt{x}$:

$$f(y) = \frac{1}{y^2} - x \tag{3.13}$$

Its derivative is:

$$f'(y) = \frac{-2}{y^3} \tag{3.14}$$

By applying 3.13 and 3.14 to 3.10, equation 3.15 is obtained:

$$y_{i+1} = y_i + \frac{y_i}{2} - \frac{y_i^3 x}{2} \tag{3.15}$$

This equation can be applied iteratively to make y a better approximation of the inverse square root of x. In this work, only one iteration was considered.

Let isqrt_i and isqrt_o denote, respectively, the input and the output of the inverse square root hardware. The again modified algorithm, now using the inverse square root and multiplications, becomes:

```
R = 0
for k=0:N-1
        isqrt_i = h_k^H . h_k
        isqrt_o = 1/√isqrt_i
        R(k, k) = isqrt_i.isqrt_o
        q_k = h_k .isqrt_o
        for j=k+1:N-1
                R(k, j) = q_k^H . h_j
                h_j = h_j – q_k . R(k, j)
        end
end
```

Algorithm 2: MGS modified for hardware implementation

This algorithm has yet the undesired property that the input matrix **H** is altered during its execution and therefore it would need to be copied by the hardware to an internal memory. However, after the k^{th} column of **Q** is calculated, the k^{th} column of the

copy of **H** is no longer used. For this reason, also as suggested in (GOLLUB, 1996), this code can be further improved with the merge of this copy of the input matrix and **Q**. Therefore, Algorithm 3 is also functional, given that **Q** is initialized with the input matrix:

```
R = 0; Q = H
for k=0:N-1
    isqrt_i = q_k^H . q_k
    isqrt_o = 1/√isqrt_i
    R(k, k) = isqrt_i.isqrt_o
    q_k = Q(0:N-1, k).isqrt_o
    for j=k+1:N-1
        R(k, j) = q_k^H . q_j
        q_j = q_j – q_k . R(k, j)
    end
end
```

Algorithm 3: MGS with improved memory usage

3.3 Sorted QR Decomposition

The order in which the transmitted signals are evaluated can affect the performance of the sphere decoding algorithm (STUDER, 2008). This order can be modified using the sorted QR decomposition (SQRD) algorithm.

The intent of the SQRD is to sort the elements of main diagonal of **R**, $\mathbf{R}_{k,k}$, decreasingly in the order of evaluation of the sphere decoding algorithm, i.e. from the bottom right corner to the top left corner. The benefits of doing so can be seen in equation 2.18: the larger $\mathbf{R}_{k,k}$ is, the more different will the PED increases of each brother node be, as $\mathbf{R}_{k,k}$ can amplify the value of s_i and thus improve the range of the search. This effectively increases the average PED and average PED variance in the layers closer to the root, which will result in more branches reaching the sphere radius in the top layers and therefore being left out of the search.

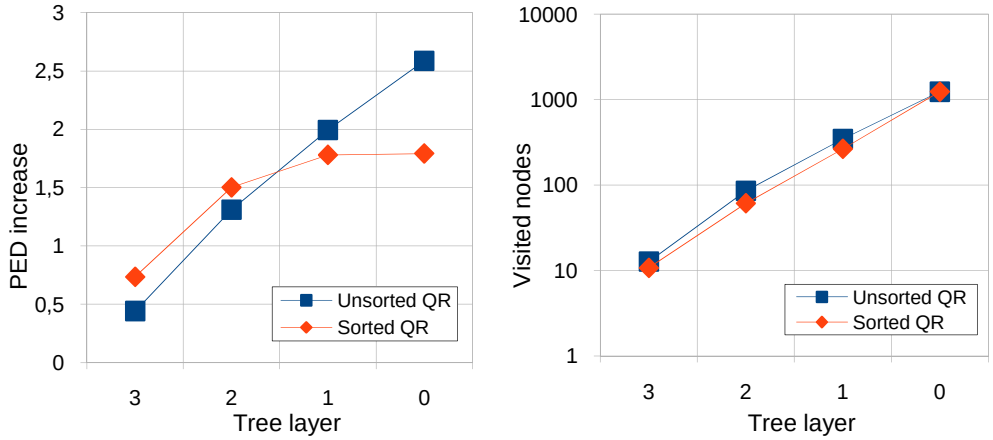

Figure 3.2: Average PED increase and average number of visited nodes per layer for a 4×4 system, with 16-QAM and a SNR of 12dB

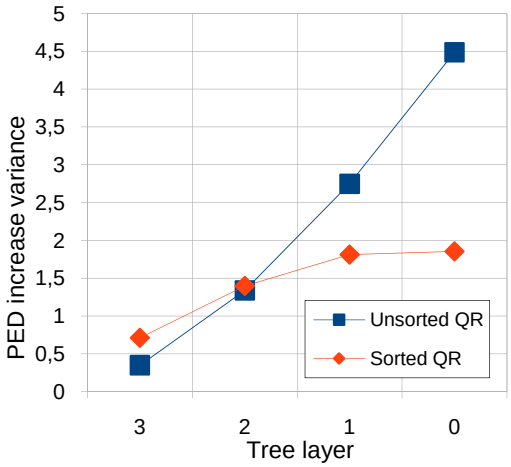

Figure 3.3: Average PED increase variance per layer for a 4×4 system, with 16-QAM and a SNR of 12dB

Also, as we get closer to the leaf nodes, the PED increases become smaller than those of the unsorted QR decomposition system. This leads to the evaluation of almost the same amount of leaf nodes, which are the ones that actually define the output. Figure 3.2 shows the average partial Euclidean distance and amount of visited nodes per tree layer for a 4×4 16-QAM system with 12dB of SNR. Figure 3.3 shows the average variance. Both variations use a constant sphere radius of 0.8 and present similar frame error rates.

The SQRD algorithm used in this work is based on the one presented in (WÜBBEN, 2001). As the elements $\mathbf{R}_{k,k}$ are calculated in the inverse order of their usage in the sphere decoder, the algorithm tries to minimize each element it calculates. Inspecting Algorithm 3, one can realize that these elements are calculated from the norm of the columns of **Q**, which was initialized with the input matrix. This SQRD algorithm chooses, before calculating each $\mathbf{R}_{k,k}$, the column of **Q** with the smallest norm and swaps it with the k^{th} column. The same exchange is done in **R** and **p**, which is the permutation vector. It is initialized with 0, 1, …, N-1 and in the end of the computation it will have a record of the swaps that were done.

R = 0; Q = H
for k=0:N-1
  i = arg $\min_{j=k:N-1} \| \mathbf{q}_j \|^2$
  Exchange columns k and i in Q, R and p
  R(k, k) = $\| \mathbf{q}_k \|$
  $\mathbf{q}_k = \mathbf{q}_k / R(k, k)$
  for j=k+1:N-1
    $R(k, j) = \mathbf{q}_k^H . \mathbf{q}_j$
    $\mathbf{q}_j = \mathbf{q}_j - \mathbf{q}_k.R(k, j)$
  end
end

Algorithm 4: Sorted QR decomposition (SQRD)

It is important to note that the ordering obtained by Algorithm 4 is an approximation, rather than the exact optimal ordering. For simplicity, the algorithm is presented with the original mathematical definitions of Algorithm 1.

In Algorithm 4, however, it is clear that the addition of sorting to the algorithm is expensive. It involves calculating the squared norm (inner product with itself) of the k^{th} vector and all vectors to the right of it at each iteration, instead of only the squared norm of the k^{th} vector. In a 4×4 system, for example, this means that 6 extra squared norm calculations are to be done.

However, Algorithm 4 can be improved to calculate the norm of each column of **Q** only once, and then only update this value as the projections are subtracted in the inner loop (WÜBBEN, 2003). This is done with a norm vector, which contains the squared norm of each column of **Q**. Let *conj* denotes the complex conjugate of a number. The SQRD algorithm using the norm update method becomes:

```
R = 0; Q = H
for k=0:N-1
        norm(k) = || q_k ||^2
end
for k=0:N-1
        i = arg min_{j=k:N-1} norm(j)
        Exchange columns k and i in Q, R, p and norm
        R(k, k) = || q_k ||
        q_k = q_k / R(k, k)
        for j=k+1:N-1
                R(k, j) = q_k^H . q_j
                q_j = q_j – q_k.R(k, j)
                norm(j) = norm(j) – conj(R(k, j)).R(k, j)
        end
end
```

Algorithm 5: Improved sorted QR decomposition (SQRD)

With this improvement, the amount of inner products to be calculated remains the same as in the unsorted QR decomposition, and the complexity increase is reduced to the ordering and norm updating steps.

The **p** vector obtained after the computation of the SQRD is necessary to reorder the output, so that it matches the values expected by the channel decoder, as shown in Figure 3.4. The a-priori information obtained from the channel decoder also needs to be reordered to match the new sorting of the transmit antennas.

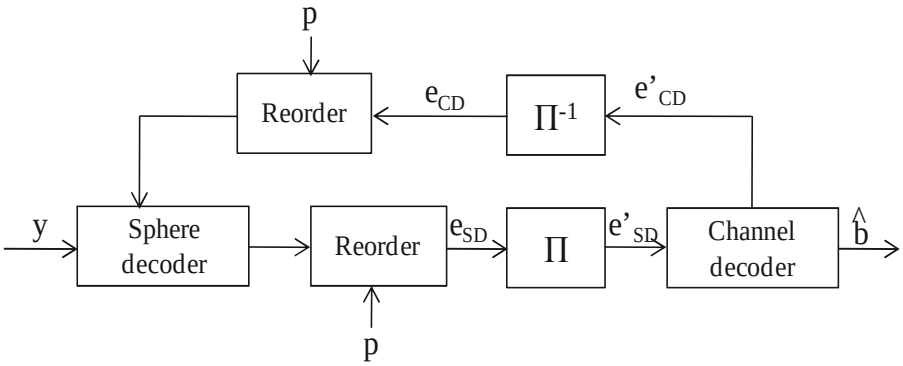

Figure 3.4: Receiver modified for SQRD usage

3.4 MMSE Pre-processing

As the SQRD algorithm, the minimum mean squared error (MMSE) pre-processing was initially developed to be used with linear decoders (WÜBBEN, 2003). For those detectors, as the name suggests, it is used to reduce the probability of errors, as is the SQRD algorithm. The same modification happens when this technique is brought to sphere decoders, i.e. it can be used to reduce the amount of visited nodes (MENNENGA, 2009). It can also be coupled with the SQRD algorithm, thus achieving further complexity reductions. Also as occurs with SQRD, the MMSE pre-processing reduces the average amount of visited nodes by increasing the average PED and the average PED increase variance in the top layers, allowing bad branches to be pruned earlier in the tree, as shown in Figures 3.5 and 3.6.

The MMSE pre-processing is done using an extended $(N_T+N_R) \times N_T$ matrix as the input. In our particular case, the input matrix dimensions become 2N×N. Let I_N denote the N-order identity matrix and σ denote the standard deviation of the noise vector. The modified matrices then become:

$$H' = \begin{bmatrix} H \\ \sigma.I_N \end{bmatrix} = Q'R' = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R' \tag{3.16}$$

Where $\mathbf{Q_1}$ and $\mathbf{Q_2}$ are square matrices of order N.

It is important to note at this point that, from equation 3.16, $\sigma\mathbf{I_N}=\mathbf{Q_2R'}$. From this comes that:

$$R'^{-1} = \frac{1}{\sigma} Q_2 \tag{3.17}$$

This means that $\mathbf{Q_2}$ is the scaled inverse of $\mathbf{R'}$. Since $\mathbf{R'}$ is an upper-triangular matrix, so is $\mathbf{Q_2}$. This property will be used during the hardware architecture design.

The value of σ for the used channel model (equation 2.2) is associated with the signal to noise ratio. Since the **n** vector has a standard deviation of 1 and it is scaled by the inverse square root of the SNR, the noise standard deviation is:

$$\sigma = \frac{1}{\sqrt{SNR}} \tag{3.18}$$

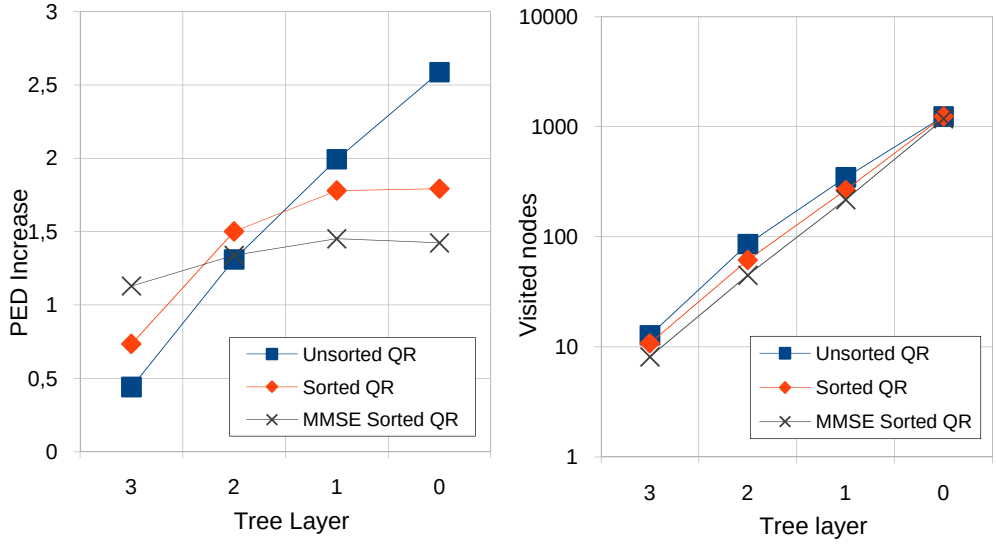

Figure 3.5: Average PED increase and average number of visited nodes per layer for a 4×4 system, with 16-QAM and a SNR of 12dB

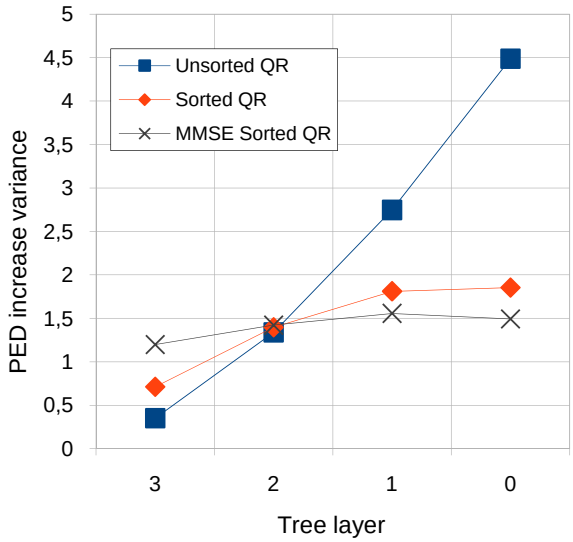

Figure 3.6: Average PED increase variance per layer for a 4×4 system, with 16-QAM and a SNR of 12dB

Aside from the different size and initialization of **Q**, the algorithm to perform the MMSE-SQRD is almost equal to that of the original SQRD. The main difference is that not all rows of **Q** are swapped, only the first N+i. In order to match the extended output matrix **Q'**, the received vector **y** also needs to be modified, becoming **y'** = $[\mathbf{y}\ 0_N]^T$, where 0_N denotes the N-order zero vector. This way, the resulting vector becomes:

$$\hat{y}' = Q'^{H}.y' = Q_1^{H}.y \tag{3.19}$$

The $\hat{y}'$ vector has the same dimension of the original $\hat{y}$ vector and is then used in the same way by the sphere decoder.

The algorithm to perform the SQRD in the modified matrices is:

$R = 0$; $Q_1 = H$; $Q_2 = \sigma I_N$; $p = [0, 1, \ldots, N-1]$

for k=0:N-1

$norm(k) = \| \mathbf{q}_k \|^2$

end

for k=0:N-1

$i = \arg \min_{j=k:N-1} norm(j)$

Exchange columns k and i in R, p and norm and the first N+i rows of Q

$R(k, k) = \| \mathbf{q}_k \|$

$\mathbf{q}_k = \mathbf{q}_k / R(k, k)$

for j=k+1:N-1

$R(k, j) = \mathbf{q}_k^H . \mathbf{q}_j$

$\mathbf{q}_j = \mathbf{q}_j - \mathbf{q}_k.R(k, j)$

$norm(j) = norm(j) - conj(R(k, j)).R(k, j)$

end

end

Algorithm 6: Sorted QR decomposition with MMSE pre-processing

3.4.1 Bias subtraction

As a result from the mentioned alterations in the matrices and vectors, the MMSE pre-processing introduces a bias in the distance calculation metrics (MENNENGA, 2009):

$$\left\| y' - H's \right\|^2 = \left\| y - Hs \right\|^2 + \sigma^2 \left\| s \right\|^2 \tag{3.20}$$

This bias must be subtracted from the distance calculated by the sphere decoder in order to avoid an increase in the error rates. There are many different ways to do so, and in this work, three different approaches were evaluated.

Aside from the bias, the distances themselves are modified by the MMSE algorithm, which means that the optimal values for the sphere radius that are used with the other QRD algorithms may not apply directly in this case, and are also likely to be different for each norm subtraction method. Further analysis and simulation results on this matter can be found in chapter 4.

3.4.1.1 Late subtraction

One of the possible approaches is to subtract the bias as late as possible. This means that the sphere decoding is executed completely ignoring the bias and then it is subtracted only when the LLRs are calculated:

$$\Lambda_j = (\min 1_j - bias 1_j) - (\min 0_j - bias 0_j) \tag{3.21}$$

This approach has, however, some disadvantages. First, it is necessary to keep track of the bias that was introduced in each bit when the minimum distance for it being 0 or

1 was found, which requires two extra vectors. Second, the minima vectors update is done with biased distances. Since the bias is the norm of the s vector, multiplied by σ, this creates a preference for points that are closer to the origin of the QAM constellation, which will have reflections in the resultant error rates.

3.4.1.2 Early subtraction

As the distances are calculated recursively by the algorithm, so can the bias subtraction be done. The module of the s vector can be separated into the module of each individual symbol, and this value can be subtracted directly as the PEDs are calculated. The unbiased PEDs are calculated with:

$$\left|e_i(s^{(i)})\right|^2 = \left|\hat{y}_i - \sum_{j=i}^{N-1} R_{ij} s_j\right|^2 + \sum_k LLR(a_k, b_k) - \sigma^2 \left|s_i\right|^2 \tag{3.22}$$

The advantages of this approach are that, since the subtraction is done as early as possible, the system is entirely bias-free. Also, it requires no extra storage elements. However, it has a tendency of visiting more nodes than the other approaches, since the bias is always positive and therefore the PEDs obtained with this method are always smaller. On the other hand, this may mean that smaller sphere radii are acceptable when using early subtraction.

3.4.1.3 Intermediate subtraction

This method is an attempt to take the main advantages of the two others. In order to avoid a preference for points closer to the origin, the bias must be subtracted before the minima vectors are updated. However, to reduce the amount of visited nodes, the bias must be subtracted after the PEDs are compared to the sphere radius. With these two restrictions, the only sphere decoder step to perform the bias subtractions is when a leaf node is reached, before the minima update.

To do so, whenever a leaf node is reached, the sphere decoder must calculate the bias associated with the symbols in the path towards the tree root and then subtract it from the distance associated with the point that was reached. The unbiased distance is then used to update the minima vectors.

This approach requires no extra storage elements when using a constant sphere radius. However, when sphere shrinking is used, the minima vectors are used to dynamically determine the sphere radius. Since the PEDs are still biased and the minima are not, this causes the comparison of a biased distance with an unbiased sphere radius, resulting in the early exclusion of many relevant branches from the tree. To avoid this problem, auxiliary vectors are necessary to keep the biased minima vectors. These vectors are used to determine the sphere radius and the unbiased ones to calculate the LLRs in the end of the execution.

4 SIMULATION CHAIN AND RESULTS

The model described in chapter 2 was entirely coded as a C++ simulation model, using the IT++ library for vector and matrix handling. The main purpose of this model was to analyse the effects that the changes in the QR decomposition algorithm would have in the resultant FER and computational effort.

To evaluate this changes, the QR decomposition algorithm was coded with the ac_fixed and ac_complex data types, provided with Mentor Graphics' Catapult. These types allow the definition of the amount of integer and fractional bits, signedness, rounding and saturation. The rest of the chain remained with floating point precision. With this approach it is possible to evaluate the isolated effects of the fixed point precision in the QR decomposition, since the rest of the system remains equal.

In most graphs, the result with floating point QR decomposition is also plotted, for comparison reasons. The format x.y denotes x integer bits, including the sign bit, and y fractional bits for fixed point implementations.

4.1 Simulation parameters

There are many different parameters that define each simulation. Regarding the communication chain as a whole, there are the amount of different QAM symbols (and the according amount of bits represented by each symbol), the size of the frame word, the amount of transmit and receive antennas (as mentioned, these two numbers are considered always the same in this work), the signal to noise ratios to be simulated and the amount big loop iterations to be executed. Considering simulation-only parameters, the most important is used to define the end of the simulation. The parameter used was a limit of frame errors, typically 50, 100 or 150. However, aside from this frame error limit, there was also a limit of frames sent, which in all cases was 100000.

Another important fact is that early-stopping is used to accelerate the simulation. This means that if the output is correct before the last big loop iteration, the process is finished and the chain moves to the next frame.

Concerning the sphere decoder itself, the main parameters are the sphere radius to be used and the usage of sphere shrinking, as discussed in chapter 2.

As for the QR decomposition, the most important parameters are the amount of integer and fractional bits and the usage or not of saturation and rounding. Another important parameter is the usage or not of a Newton-Raphson iteration at the output of the approximation inverse square root hardware. This has a huge influence in the resulting FER.

Some of these parameters are kept equal in all simulations, since they are not a part of the scope of analysis. It is used 5 big loop iterations and an initial sphere radius of

0.8, which is sufficient to allow a small loss (SIMHA S, 2009), unless stated otherwise. Also the equation for addition of a-priori information is not changed. All fixed point operations were executed with rounding and saturation. The use of rounding means that the quantization of the input matrix and the reduction of the output of the multipliers consider the most significant bit left out of the result. If this bit is 1, then 1 is added to the least significant bit of the result. The use of saturation means that, whenever an overflow would happen, the result is replaced with the largest number possible in the used representation format. Unless stated otherwise, the values in table 4.1 are the ones used in the simulations.

Table 4.1: Default parameters for simulations

Parameter	Value
Initial sphere radius	0.8
M (QAM Symbols)	16
Frame word size	994 bits
Big loop iterations	5
Sent frames limit	100000

4.2 Simulation results

4.2.1 Effect of the quantization of the output

The effect of a fixed point output can be analysed isolated from the effect of the whole algorithm running with fixed point precision. This allows the determination of how many bits are needed in the output. The full floating point algorithm was executed and then the output's precision was limited according to different amounts of bits. However, these amounts are not necessarily related to the amount of bits necessary in the internal calculations.

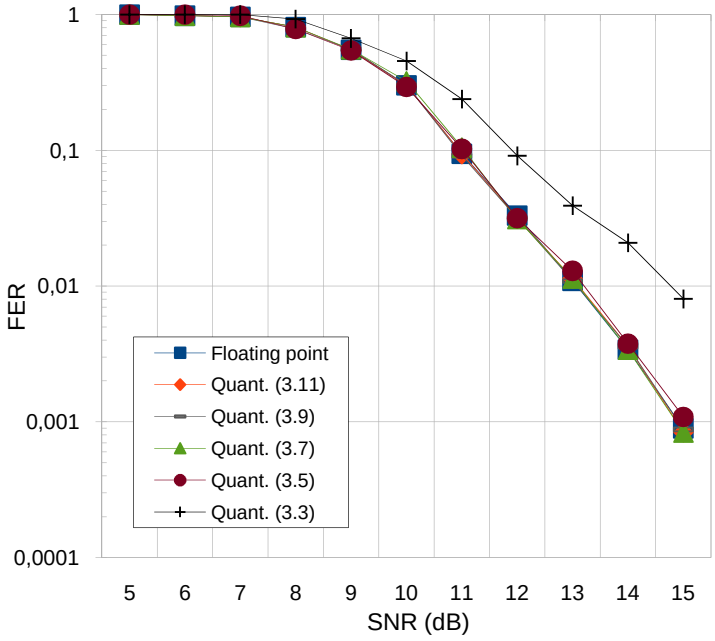

Figure 4.1: Effect of the quantization of the output in a 4×4 antennas system

Table 4.2: Parameters for simulations in Figure 4.1

Parameter	Value
Number of antennas	4
Frame errors limit	50

In Figure 4.1 it is possible to see that 3 integer bits and 5 fractional bits are enough to represent the output without significant increase in the FER, compared to the floating point output. However, when the number of fractional bits is further reduced to 3, this increase becomes much greater.

4.2.2 Effect of a Newton-Raphson iteration

The application of Newton-Raphson iterations is a well known method for improving the approximation of a function. One single iteration is used here to improve the output of the inverse square root approximation hardware. Figure 4.2 shows the FER gain resultant from this, using a fixed point unsorted QR decomposition, compared to the floating point version.

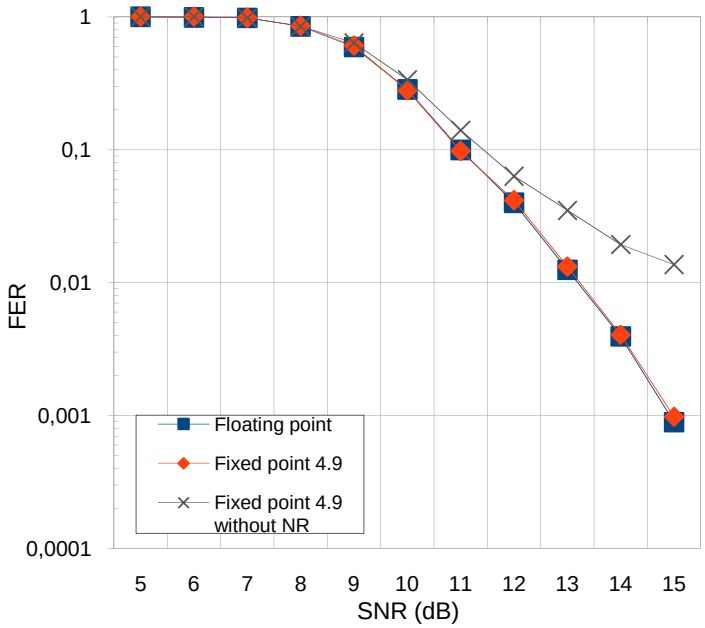

Figure 4.2: FER improvement due the use of one Newton-Raphson iteration

Table 4.3: Parameters for simulations in Figure 4.2

Parameter	Value
Number of antennas	4
Number format	4.9
Frame errors limit	100

As the FER is significantly increased when the Newton-Raphson iteration is not used, all further simulations using fixed point formats consider that this optimization is activated.

4.2.3 Amount of bits necessary for 2×2 and 4×4 systems

As the system to be developed requires satisfactory performance with both 2×2 and 4×4 antennas, one must analyse the minimum amount of bits required for the integer and fractional parts of both systems. Sections *4.2.3.1* to *4.2.3.4* analyse separately how many bits are required for each part of both system configurations, considering 16-QAM modulation. Section *4.2.3.5* analyses the required precision for a system targeting both configurations.

4.2.3.1 Amount of fractional bits for 4×4 antennas

To obtain the minimum amount of fractional bits necessary to introduce only a tolerable FER in a 4×4 antennas system, different formats were tested.

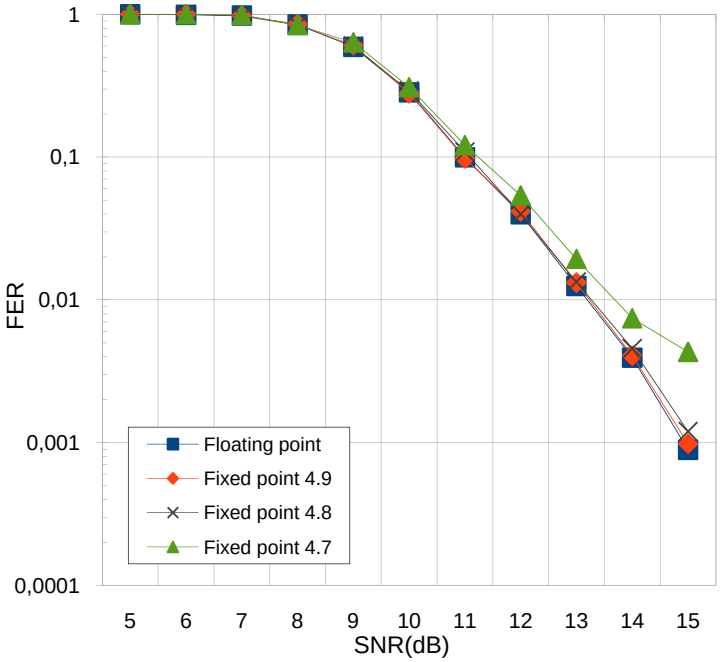

Figure 4.3: Different amounts of fractional bits in a 4×4 antennas system

Table 4.4: Parameters for simulations in Figure 4.3

Parameter	Value
Number of antennas	4
Frame errors limit	100

Simulations started with 9 fractional bits, value which was further decreased until the introduced FER became significant. As can be seen in Figure 4.3, 8 fractional bits are enough to introduce little increase to the FER.

4.2.3.2 Amount of integer bits for 4×4 antennas

As was done for the fractional bits, different amounts of integer bits were tested decreasingly until the introduced FER became too large.

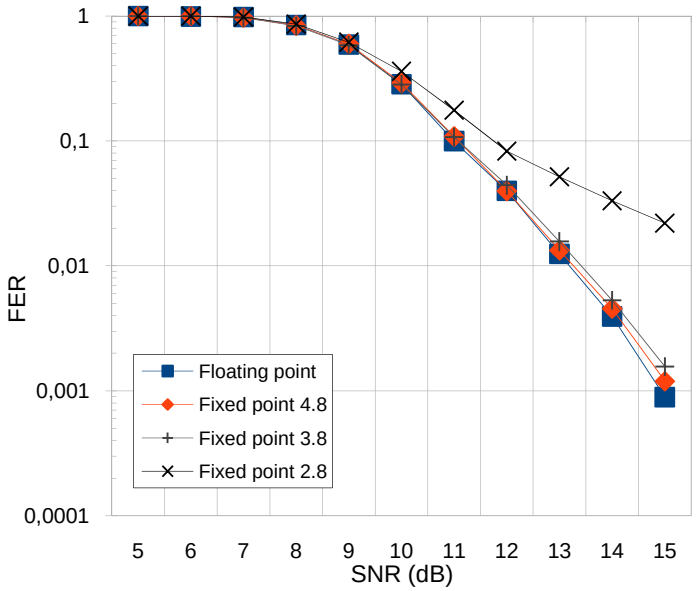

Figure 4.4: Different amounts of integer bits in a 4×4 antennas system

Table 4.5: Parameters for simulations in Figure 4.4

Parameter	Value
Number of antennas	4
Frame errors limit	100

As can be seen in Figure 4.4, the introduced FER with 3 integer bits could still be considered tolerable. With 2 integer bits, the FER introduced is too large for any real application.

4.2.3.3 Amount of fractional bits for 2×2 antennas

As for 4×4 antennas, the analysis for the optimal fixed point format was done for a 2×2 antennas system.

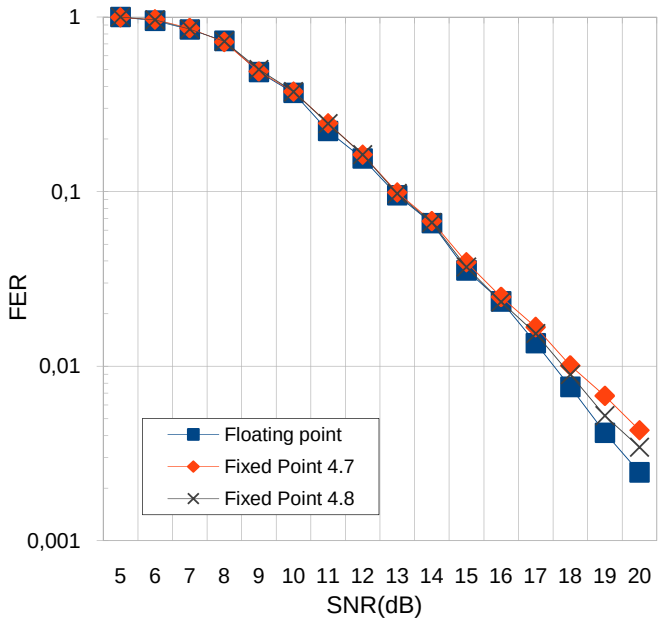

Figure 4.5: Different amounts of fractional bits in a 2×2 antennas system

Table 4.6: Parameters for simulations in Figure 4.5

Parameter	Value
Number of antennas	2
Frame errors limit	150

As shown in Figure 4.5, the system exhibits satisfactory behaviour with 8 fractional bits. The degradation with 7 fractional bits can be considered tolerable, depending on the application.

4.2.3.4 Amount of integer bits for 2×2 antennas

The same approach was applied to the amount of integer bits in a 2×2 antennas system.

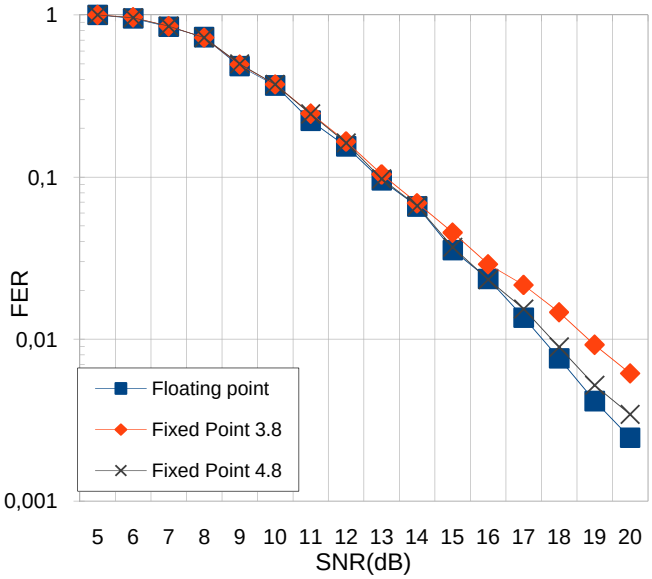

Figure 4.6: Different amounts of integer bits in a 2×2 antennas system

Table 4.7: Parameters for simulations in Figure 4.6

Parameter	Value
Number of antennas	2
Frame errors limit	150

Figure 4.6 shows that it is required to have 4 integer bits in a 2×2 antennas system in order to not introduce a large degradation, compared to the floating point implementation.

4.2.3.5 Total amount of bits

For the 4×4 system, it was determined that 3 integer and 8 fractional bits are enough. However, for the 2×2 system, 4 integer and 7 or 8 (according to application specifications) fractional bits are required. Therefore, a QR decomposition hardware that is intended for both systems should have 4 integer and 8 fractional bits to ensure a tolerable loss in precision for all cases. It is important to emphasize that this results are valid specifically for the 16-QAM modulation used. For higher order constellations, more bits may be required.

4.2.4 Effects of the sorted QR decomposition and sphere shrinking

The intent of the SQRD algorithm is to reduce the amount of visited nodes, as is the intent of sphere shrinking (SS). For this reason, in the following sections, the average amount of visited nodes is also plotted. These graphs consider only intermediate nodes the lie inside the sphere. The FER graphs are also plotted to analyse the effects of this alterations, since they may introduce significant increase in the loss, due to the fact that the a-priori information can be negative.

The combination of these techniques was also simulated, i.e. SQRD with SS and with ordered sphere shrinking (OSS). These algorithms are expected to combine their gains in the amount of visited nodes, especially when SQRD is used with OSS, as the increase of the variance in the top layers allows very good branches to be found very quickly, further accelerating the shrinkage of the sphere. The simulations in this section use the original SQRD algorithm (Algorithm 4).

4.2.4.1 Results for 4×4 antennas

All combinations between SQRD, SS, OSS and CS (constant sphere) were simulated with floating point precision at first, to analyse the effects of these combinations without the errors introduced by the fixed point quantization. The results with 2×2 antennas and also with the 4.8 fixed point format are similar and can be found in the Appendix.

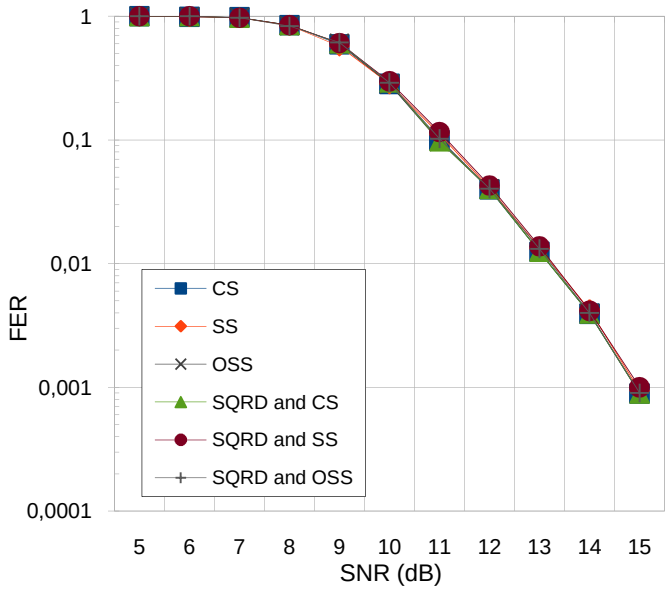

Figure 4.7: FER for different algorithms with floating point in a 4×4 antennas system

Figure 4.7 shows that there was no significant change in the frame error rate with any combination of the algorithms. This confirms the fact that the possible negative a-priori information can be neglected when using sphere shrinking. As for the average amount of visited nodes in Figure 4.8, the results confirm that the usage of sorted QR decomposition have significant impact. Also, sphere shrinking and ordered sphere shrinking can further reduce this number, and this reduction effectively stacks with that of the SQRD.

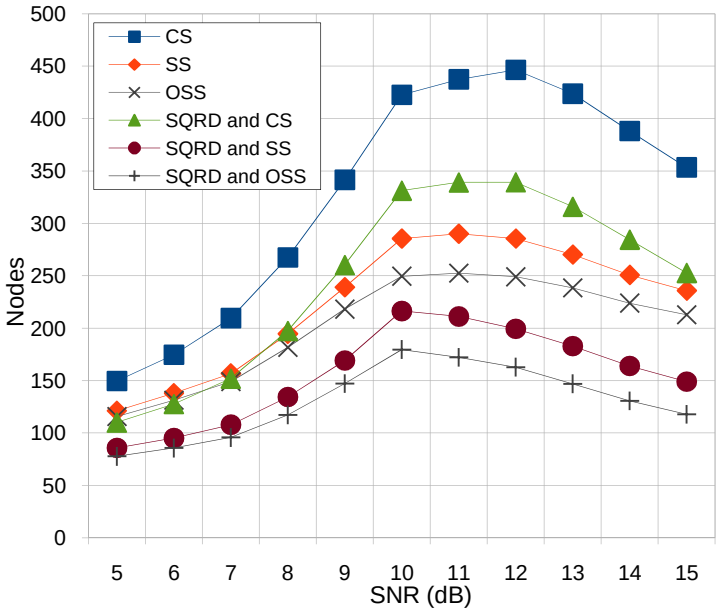

Figure 4.8: Average amount of visited nodes with floating point and 4×4 antennas

Table 4.8: Parameters for simulations in Figures 4.7 and 4.8

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Floating point

4.2.5 Effects of the usage of the norm update method

Algorithm 5, presented in chapter 3, reduces the complexity increase caused by the sorting of the columns in the input matrix by updating the norms contained in the norm vector instead of recalculating them at each iteration. This alteration, however, has side effects in the precision of the output when dealing with fixed point representations.

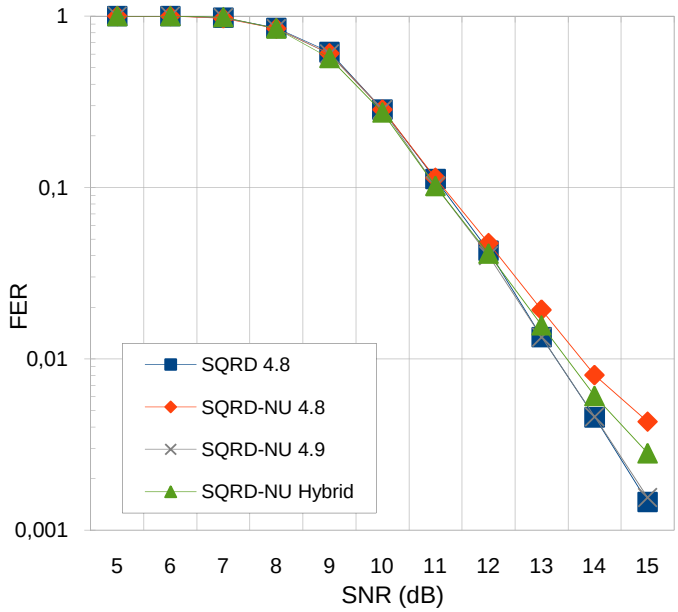

Figure 4.9: FER for different number formats using the norm update (NU) technique

Table 4.9: Parameters for simulations in Figure 4.9

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Fixed point

Figure 4.9 shows that the 4.8 bits format is indeed no longer sufficient to accurately perform the algorithm. On the other hand, it also shows that the addition of one single fractional bit eliminates this issue, reaching practically the same FER of the original SQRD algorithm. The necessity of this extra bit is due to the accumulated error in the norm vector, since the norm update used adds more quantization error at each iteration, thus having great effects especially in the last columns calculated. For this reason, a hybrid architecture was also tested, having 9 fractional bits only in the norm vector and in the norm update path. The results, however, were only intermediary, not close enough to the ones obtained with a full 4.9 hardware. Hence, and also due to the increased complexity of dealing with different representation formats in the same hardware, this option is excluded from further analysis.

This problem is not observed in 2×2 antennas systems, since the norm update hardware is used only once and the quantization error accumulated does not become significant.

4.2.6 Comparison of different bias subtraction methods for MMSE

In order to eliminate (or reduce) the increase in the error rates caused by the usage of MMSE pre-processing, different methods for subtracting the introduced bias in the metrics were presented: namely, late subtraction (LS), early subtraction (ES) and intermediate subtraction (IS). In this section, the FER and average amount of visited nodes with each method is compared. The results obtained with the original SQRD are also plotted for comparison.

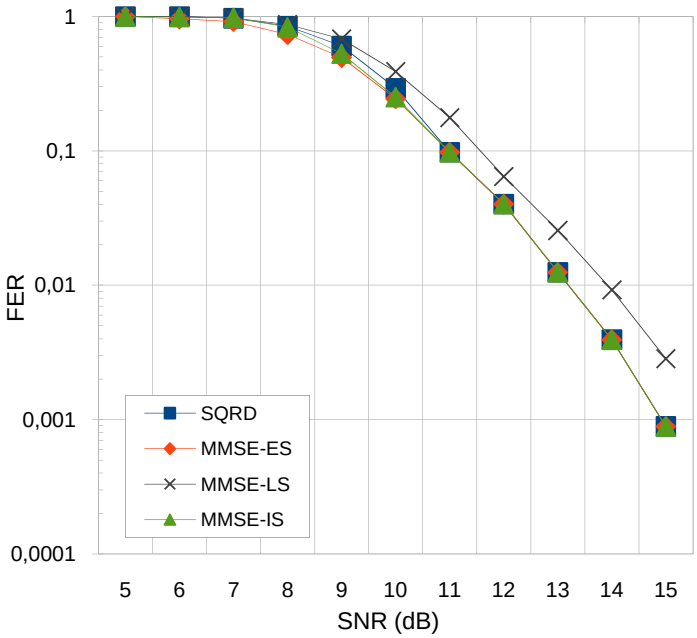

Figure 4.10: FER for different bias subtraction techniques

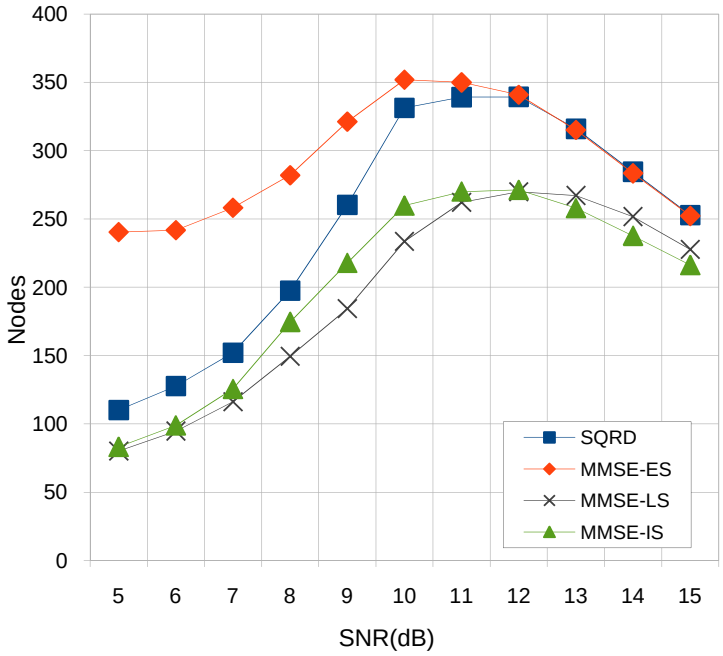

Figure 4.11: Average amount of visited nodes for different bias subtraction techniques

Table 4.10: Parameters for simulations in Figures 4.10 and 4.11

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Floating point

Figure 4.10 shows that, as expected, the late subtraction method is not suitable for our case, due to the significant increase in the frame error rate. For this reason, it is excluded from further simulations. Also, as can be seen in Figure 4.11, the early subtraction method presents no gain in the amount of visited nodes, when compared to the original SQRD algorithm. On the other hand, the intermediate subtraction method has almost identical FER when compared to the original SQRD and the MMSE-ES algorithms and still achieves an amount of visited nodes which is comparable to that of the MMSE-LS method. All these simulations, however, consider a constant sphere radius of 0.8.

4.2.7 Reduced sphere radii for MMSE-SQRD

Since the distance metrics are modified by the MMSE-SQRD algorithm, new optimal values for the sphere radius, i.e. the smallest value that causes no significant increase in the error rates, are to be found. Also, it is likely that each subtraction method has a different optimal sphere radius.

Figure 4.12 shows the FER associated with different constant sphere radii, using the early subtraction method and Figure 4.13 shows the amount of visited nodes associated with each of these radii, considering a constant sphere.

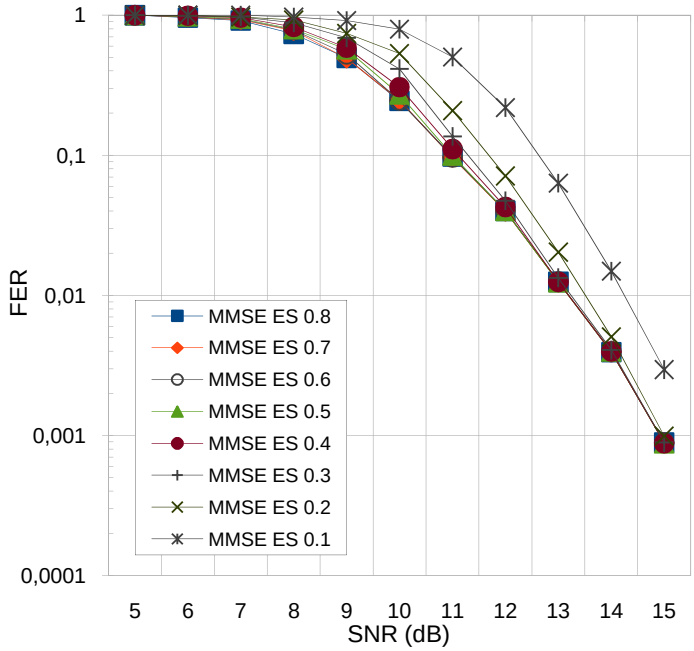

Figure 4.12: FER for different sphere radii in using early bias subtraction

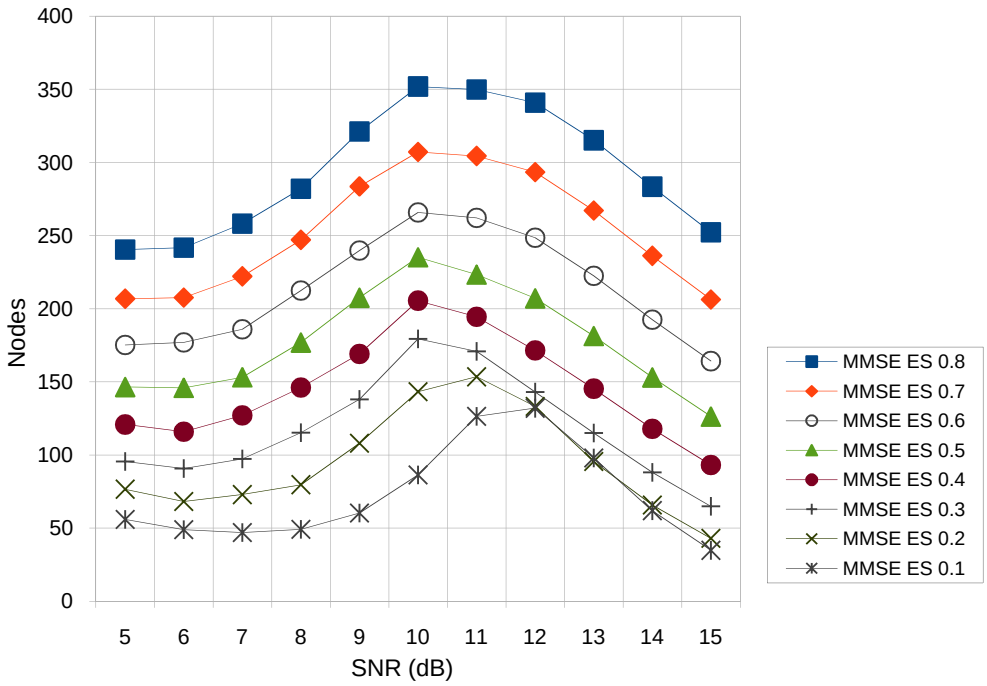

Figure 4.13: Visited nodes for different sphere radii in using early bias subtraction

Table 4.11: Parameters for simulations in Figures 4.12 and 4.13

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Floating point
Bias subtraction method	Early subtraction

Figure 4.12 shows that 0.4 is the smallest radius that ensures no significant FER increase for all simulated SNRs. For higher SNRs, however, much smaller radii are acceptable. In Figure 4.13 it is visible that the reduction obtained in the average amount of visited nodes is very significant when the reduced radii are used.

The same analysis was done for the intermediate bias subtraction.

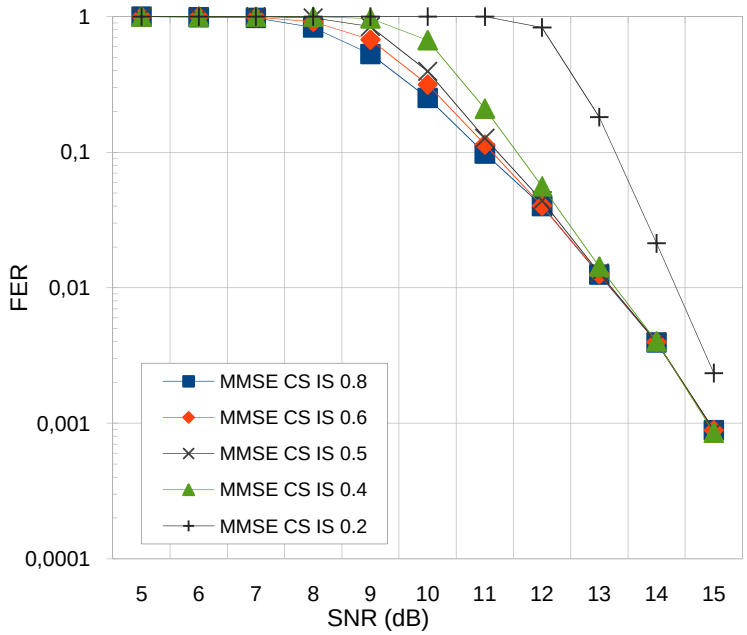

Figure 4.14: FER for different sphere radii in using intermediate bias subtraction

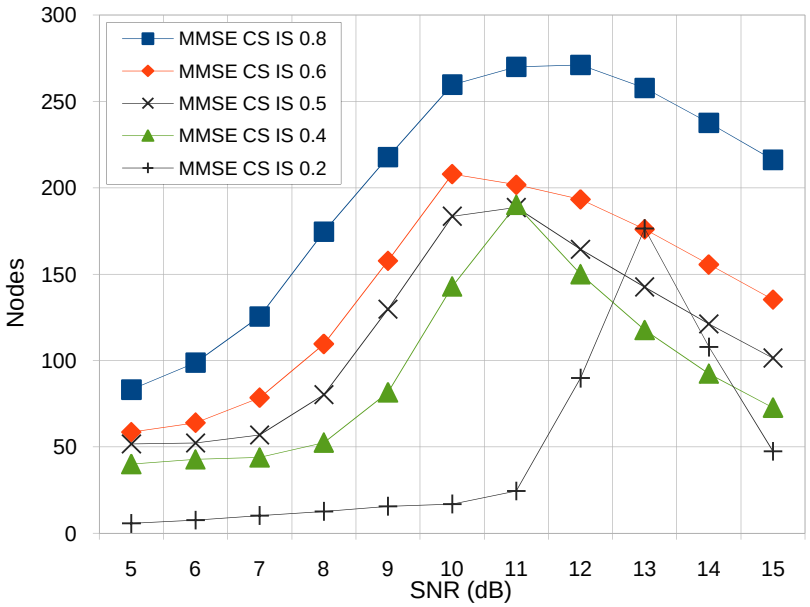

Figure 4.15: Visited nodes for different sphere radii in using intermediate bias subtraction

Table 4.12: Parameters for simulations in Figures 4.14 and 4.15

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Floating point
Bias subtraction method	Intermediate subtraction

Also when using intermediate subtraction, a smaller sphere is acceptable when dealing with higher SNRs. However, considering a constant sphere for all SNRs, the minimum value would be around 0.6, as shown in Figure 4.14, which is the value used for further comparison with the early subtraction method. In Figure 4.15 the associated visited nodes can be seen for each radius.

Figure 4.15 also shows some points in which smaller spheres visit averagely more nodes than larger spheres. This is due to the early stopping: when a larger sphere is used, more frames are correctly decoded in the initial big loop iterations, and this is also reflected in the large difference between FERs. Since the trees built in these iterations tend to have much less nodes then in the last ones, the algorithm can actually visit averagely more nodes with a smaller sphere, in some very specific points for some configurations.

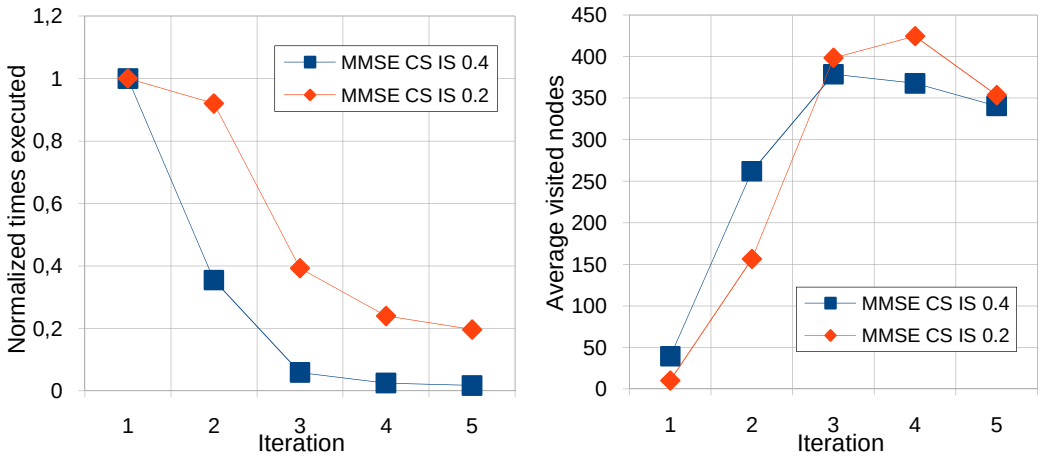

Figure 4.16: Normalized executions and average visited nodes for each big loop iteration with 13dB and different constant sphere radii

Simulations in Figure 4.16 illustrate the variation in the amount of nodes per iteration, and the frequency each iteration is reached. It considers a constant sphere, with MMSE pre-processing and intermediate bias subtraction, in the specific 13dB SNR point. The amount of executions of each big loop iteration is normalized to the amount of frames sent. Also, it is important to emphasize that, even though perfect early stopping in impossible, good approximations can be used in real systems (GILBERT, 2003).

4.2.8 Comparison between ES and IS with reduced sphere

In section 4.2.7 it was shown that both early and intermediate bias subtraction methods allow a reduction in the sphere radius without significant increase in the error rates. The minimum sphere radii were determined to be 0.4 for early subtraction and 0.6

for late subtraction, considering that the same radius is used for all SNRs. The frame error rates and average amount of visited nodes obtained with each of these systems can now be compared, as can the effects of sphere shrinking in each one of them.

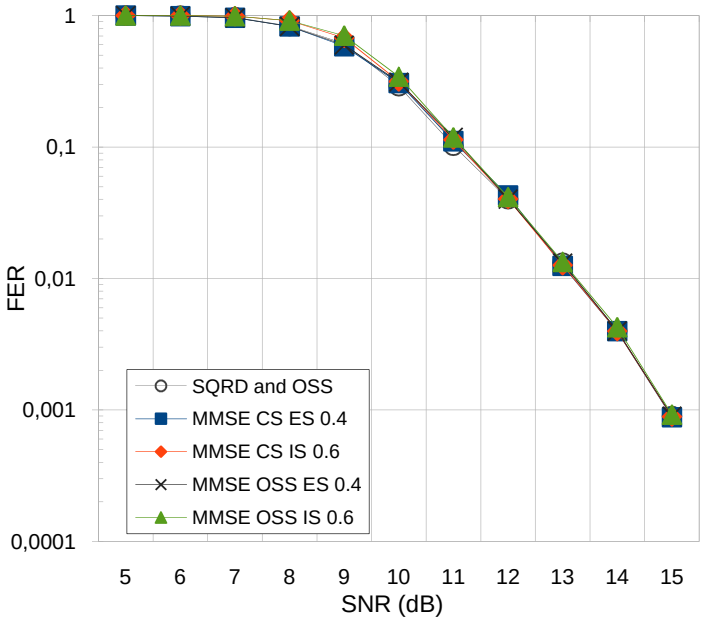

Figure 4.17: FER for each bias subtraction method with its minimum sphere radius

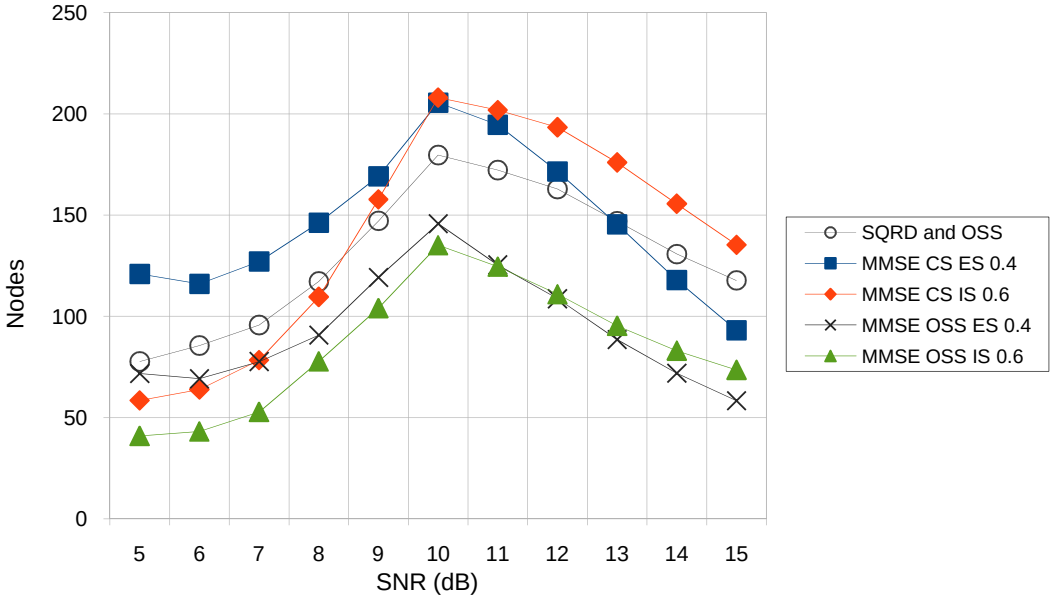

Figure 4.18: Visited nodes for each bias subtraction method with its minimum sphere radius

Table 4.13: Parameters for simulations in Figures 4.17 and 4.18

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Floating point

Figure 4.17 shows that, even with a smaller radius, the early subtraction method has a slightly better FER performance for 8 and 9dB of SNR. For the other SNRs, the FER is nearly identical. In Figure 4.18, IS shows better performance for smaller SNRs and ES for larger. The best results achieved without MMSE regarding the amount of visited nodes, which were with SQRD and OSS, are also plotted for comparison reasons. Early subtraction could outperform these results even with a constant sphere, for high SNRs. When ordered sphere shrinking is used, ES can visit less than half the amount of nodes visited by SQRD at 15dB.

4.2.9 Effects of error in the σ estimation for MMSE

Works on MMSE-SQRD so far, such as (WÜBBEN, 2003) and (MENNENGA, 2009), have considered that the receiver has perfect knowledge of noise standard deviation σ. This, however, is an unreal assumption, since the receiver will only have access to approximations of this value.

In this section we analyse how resilient to these errors are systems with MMSE-SQRD. In order to make this analysis further realistic, a full fixed point system is used, i.e. not only the QR decomposition is performed with fixed point, but also the sphere and channel decoding. The system was simulated with early subtraction and considering the noise overestimated and underestimated by 3dB. Also, for comparison reasons, the results for the floating point and full fixed point systems with constant sphere and unsorted QR decomposition are plotted.

The QR decomposition is executed with 4.8 bits, while the sphere and channel decoders use different number formats for each type of number (SIMHA S, 2009), which range from 0.6 to 4.9. The sphere radius is 0.8 for the simulations without MMSE and 0.4 for the ones with it.

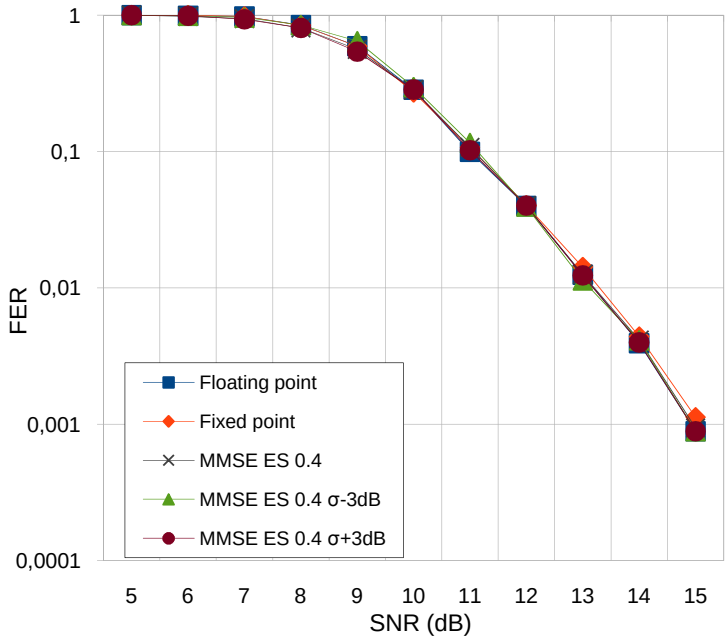

Figure 4.19: FER for full fixed point systems with sigma estimation error

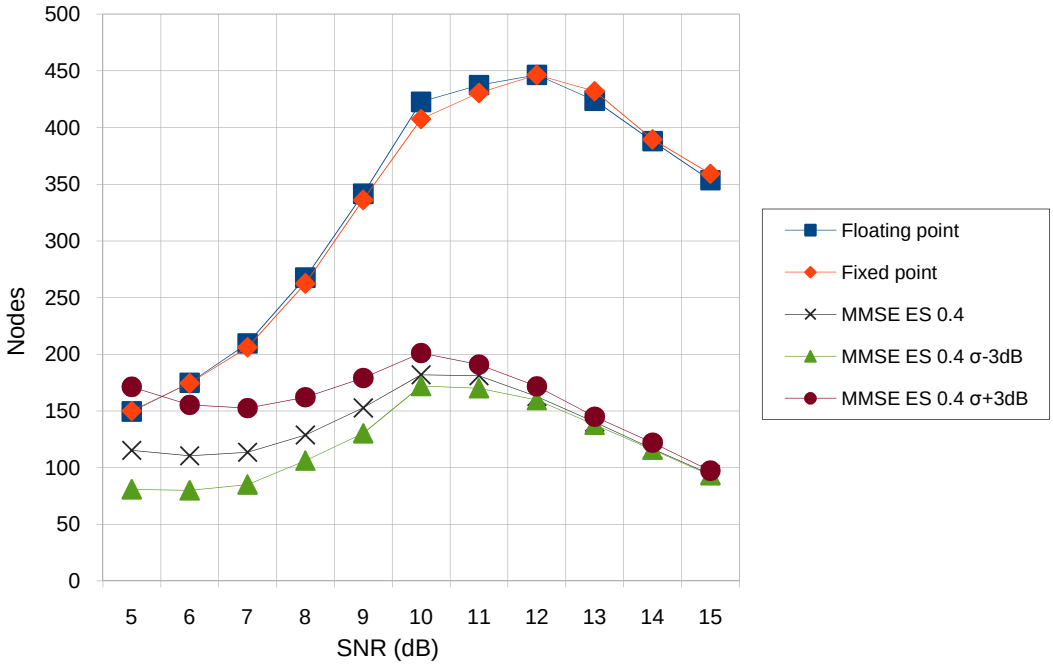

Figure 4.20: Average amount of visited nodes for full fixed point systems with sigma estimation error

Table 4.14: Parameters for simulations in Figures 4.19 and 4.20

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Full fixed point
Sphere type	Constant sphere

The system shows remarkable resilience to errors in the σ estimation, presenting almost no FER increase with the considered 3dB error, as can be seen in Figure 4.19. The amount of visited nodes, however, suffers an alteration, visiting more nodes when the noise standard deviation is overestimated and fewer nodes when it is underestimated, as shown in Figure 4.20.

Figure 4.19 also shows that a 4.8 bits format is sufficient for MMSE-SQRD fixed point implementations.

4.2.10 MMSE-SQRD with constant σ

The results in Figure 4.19 showed no significant increases in the error rates when considering an overestimation and an underestimation of 3dB. This raises the possibility of using a constant σ for large SNR intervals, which would represent a great simplification in the receptor, since it would only need a rough estimation of noise variation.

The simulations in Figures 4.21 and 4.22 use an entirely fixed point simulation chain, as described in section 4.2.9, i.e. QR decomposition with 4.8 bits and sphere and channel decoders with different formats for each number type.

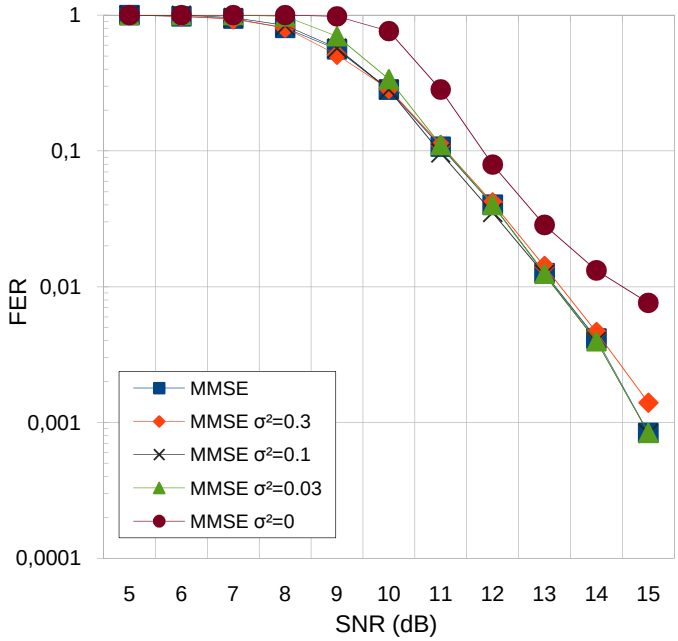

Figure 4.21: FER for MMSE-ES with different constant σ values

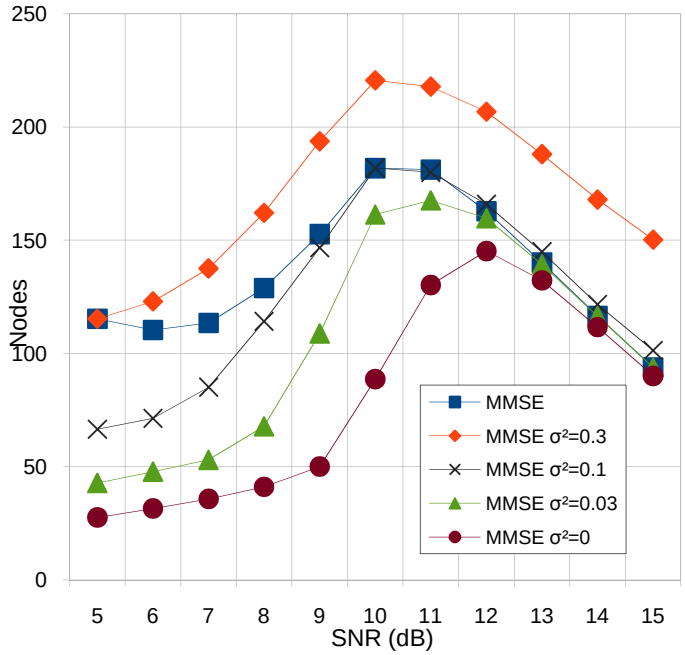

Figure 4.22: Average visited nodes for MMSE-ES with different constant σ values

Table 4.15: Parameters for simulations in Figures 4.21 and 4.22

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Full fixed point

The results show that using the σ^2 associated with an intermediate SNR value (0.1 is the value associated with a SNR of 10dB) is enough to ensure no increase in the FER and no significant increase in the amount of visited nodes in a 10dB SNR interval. The

values of 0.3 and 0.03 are rounded in the figures' legends: they are actually 0.316228 and 0.0316228, which are the values associated with 5 and 15dB of SNR, respectively. The opposite end of the graph shows a small increase the error rate in both cases. Also, the amount of visited nodes again shows that overestimating the noise causes more nodes to be visited. For all cases, all points after that which has the associated σ^2 value show increase in the average visited nodes amount.

The results with $\sigma^2=0$ are also plotted, but show significant increase in the FER, since this value would only be achieved with SNR=∞.

4.3 Simulation results summary

From the simulations in section 4.2, several conclusions were made. The Newton-Raphson improvement at the output of the inverse square root hardware was determined to be critical for a good FER performance of the fixed point QR decomposition. Also, the minimum quantization for internal representation, when using 16-QAM in the chosen SNR interval, was shown to be 4.8 bits, considering a system that runs both with 4×4 and 2×2 antennas. However, fewer bits are required at the output.

It was also shown that SQRD can successfully reduce the amount of visited nodes and that this reduction can be coupled with that obtained from sphere shrinking techniques, ordered or not. The same occurs when MMSE-SQRD is used, and it can further increase the improvements obtained from the original SQRD. The average reduction obtained by using MMSE-SQRD, when comparing the original system, is 57% less visited nodes. This value can be improved 74% when ordered sphere shrinking is used together.

Specifically for the MMSE-SQRD case, different bias subtraction methods were tested. The early and intermediate methods were the ones that successfully avoided increases in the FER. These techniques also allowed a reduction in the initial sphere radius, with the intermediate subtraction method visiting fewer nodes then the early one for lower SNRs but more for the higher ones. Also, it was shown that MMSE-SQRD systems are highly tolerant to variations in the value of σ, presenting no significant changes in the FER when a coarse approximation is used.

5 HARDWARE ARCHITECTURES

In order to implement the desired QR decomposition hardware, the presented algorithms must be translated into hardware descriptions. To do so, the first step is to come up with data paths able to perform the required operations and control finite state machines to coordinate them. In this chapter, the required operations are presented, as well as hardware schematics to perform each of them, a top level schematic to connect all of the required blocks and finally a finite state machine to control all operations, initially for the unsorted QR decomposition. The same approach is then applied to the sorted and MMSE sorted versions.

5.1 Unsorted QR Decomposition

Taking a look to the final version presented of the algorithm, a few basic operations are visible:

R = 0; Q = H

for k=0:N-1

isqrt_i = $\mathbf{q}_k{}^H . \mathbf{q}_k$

isqrt_o = $1/\sqrt{\text{isqrt\_i}}$

R(k, k) = isqrt_i.isqrt_o

$\mathbf{q}_k$ = Q(0:N-1, k).isqrt_o

for j=k+1:N-1

R(k, j) = $\mathbf{q}_k{}^H . \mathbf{q}_j$

$\mathbf{q}_j = \mathbf{q}_j - \mathbf{q}_k . R(k, j)$

end

end

It is required to have a block able to perform the *inner product* of two complex vectors, a block to compute the *inverse square root* of a real scalar, one *multiplier* for two real scalars, a *vector multiplier*, with one complex vector and one complex scalar as operands, and a *vector subtracter*, with two complex vectors as operands. To build such blocks other basic structures are required, such as *complex multipliers* and *adders/subtracters*. Storage elements for the matrices are also required.

5.1.1 Complex multiplier

The multiplication of two complex numbers is an operation that requires the use of the distributivity of multiplication over addition. The full expression for two complex numbers a and b becomes:

$$(a_{\mathrm{Re}} + a_{\mathrm{Im}} i)(b_{\mathrm{Re}} + b_{\mathrm{Im}} i) = (a_{\mathrm{Re}} b_{\mathrm{Re}} - a_{\mathrm{Im}} b_{\mathrm{Im}}) + (a_{\mathrm{Re}} b_{\mathrm{Im}} + a_{\mathrm{Im}} b_{\mathrm{Re}}) i \quad (5.1)$$

Figure 5.1 shows a simple hardware scheme to explore the possible parallelism available in this operation.

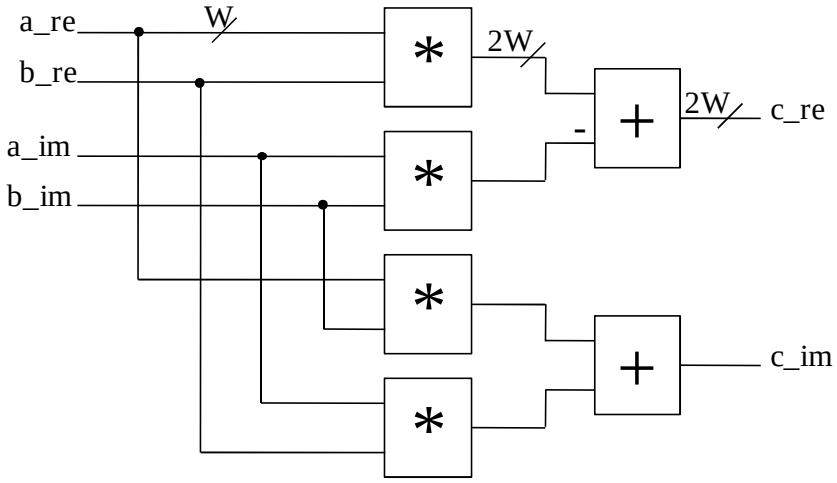

Figure 5.1: Complex multiplier schematic

The operands and results sizes are also important at this point. If each part of the operands have W bits, each part of the result should also have W bits, since it is going to be used in subsequent parts of the algorithm. Hence, rounding and saturation should be used to discard, with reduced the loss in precision, the extra bits obtained from the multiplication.

Considering a fixed point number with X integer bits and Y fractional bits (X+Y=W), the result of its multiplication with another number in the same format has 2X integer and 2Y fractional bits. To fit the output in the same format of the inputs, X integer and Y fractional bits must be removed, as shown in Figure 5.2. This should be done after the addition and subtraction in Figure 5.1, to achieve a better precision. For this reason, those two operations are done with 2W bits.

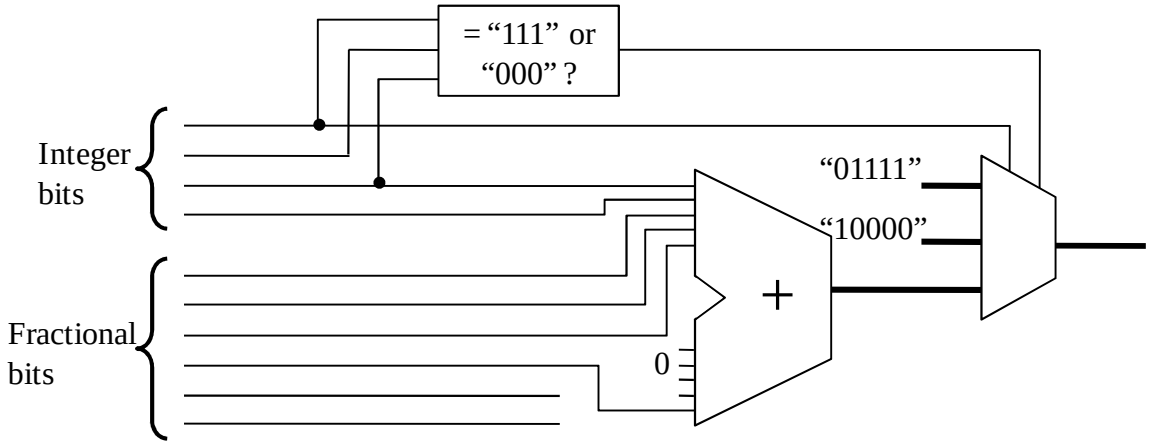

Figure 5.2: Rounding and saturation schematic for a 2.3 format

Saturation consists of replacing the output with the closest value possible whenever an overflow occurs (this can be either the largest positive or smallest negative available in the representation).

Rounding consists of adding the most significant bit removed from the factional part to the result. This effectively divides by two the maximum error caused by the removal of the Y least significant bits, when compared to simple truncation, as can be seen in Figure 5.3, where Δ is the distance between two adjacent numbers in the representation format ($1/2^Y$). While the maximum error is given by Δ when truncation is used, it is only Δ/2 when rounding is used.

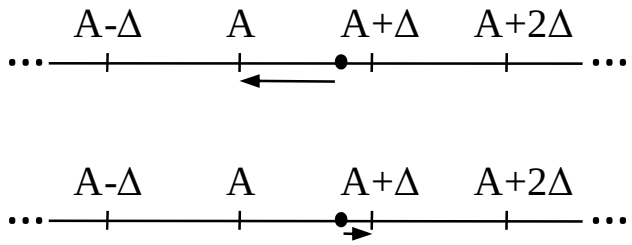

Figure 5.3: Comparison of truncation (top) and rounding (bottom)

5.1.2 Adder/Subtracter

The addition and subtraction of complex numbers is much simpler then the multiplication. The reasons are that the amount of bits produced is the same as in the input (if the carry out is ignored) and that direct associativity can be applied. The expression for two complex numbers a and b is:

$$\left(a_{\mathrm{Re}} + a_{\mathrm{Im}} i\right) + \left(b_{\mathrm{Re}} + b_{\mathrm{Im}} i\right) = \left(a_{\mathrm{Re}} + b_{\mathrm{Re}}\right) + \left(a_{\mathrm{Im}} + b_{\mathrm{Im}}\right) i \tag{5.2}$$

This means that two simple adders can be applied to perform this operation. The result, however, must still be checked for overflow and replaced by the largest positive or smallest negative available if necessary.

When an addition is calculated, an overflow can occur only if the two inputs have the same sign. In this case, an overflow happened if the sign of the output is different from that of the inputs.

In the subtraction case, however, an overflow can occur only if the two inputs have different signs. In this case, an overflow happened if the sign of the output is different from that of the first input.

It is important to note that this overflow checking must also be performed in the adders and subtracters present in the complex multipliers. In this case, it can be done in parallel to the rounding and saturation due to reduction of the bit amount, presented in Figure 5.2, provided the adders operate with the increased bit widths.

5.1.3 Inner product

The inner product (or dot product) of two complex vectors, as described by equation 3.2, is the summation of the product of the conjugate of each i^{th} element of input vector h with the i^{th} element of input vector u. In order to obtain the conjugate of the elements of the first vector, sign change blocks would be required. However, swapping the adder and the subtracter in the complex multiplier presented in Figure 5.1 gives the same result without the need of this extra block, as shown by equation 5.3. The multiplication blocks used in this section have this alteration.

$$(a_{\mathrm{Re}} - a_{\mathrm{Im}}i)(b_{\mathrm{Re}} + b_{\mathrm{Im}}i) = (a_{\mathrm{Re}}b_{\mathrm{Re}} + a_{\mathrm{Im}}b_{\mathrm{Im}}) + (a_{\mathrm{Re}}b_{\mathrm{Im}} - a_{\mathrm{Im}}b_{\mathrm{Re}})i \quad (5.3)$$

The inner product can be done in a full parallel way, as in Figure 5.5, or in a sequential manner, using only one complex multiplier and one adder, as in Figure 5.4.

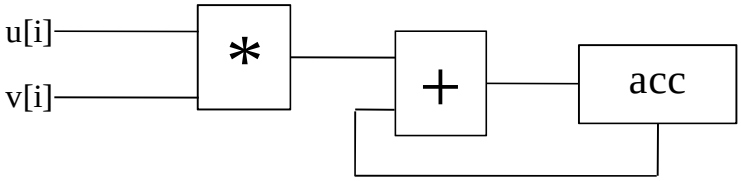

Figure 5.4: Sequential inner product block

If the multiplication, the addition and the accumulator setup times could fit in a period of the desired clock cycle then the inner product done sequentially would take N cycles. This is not likely, however, considering reasonable target frequencies. To overcome this issue, timing barriers can be added to the data path, pipelining the execution, which would then take N+D-1 cycles, where D is the pipeline depth.

The fully parallelized combinational version has N complex multipliers and N-1 complex adders. The amount of cycles required to compute the result is variable, depending on the technology used and target frequency.

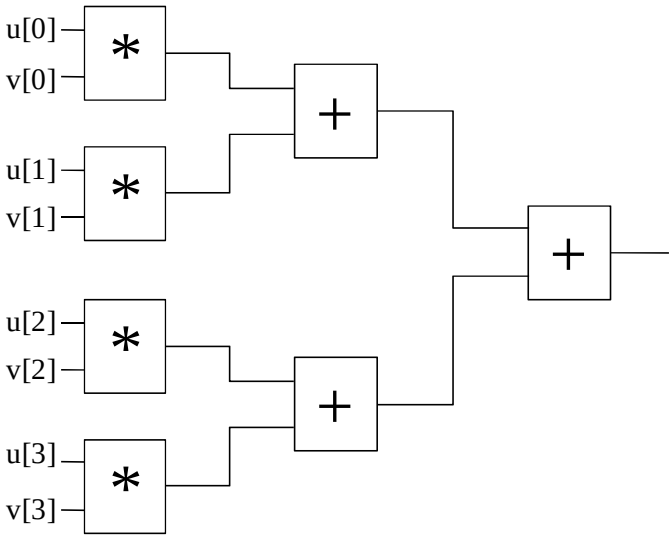

Figure 5.5: Combinational inner product block, for a N=4 system

5.1.4 Inverse square root

The alterations done in the original MGS algorithm to make it more suitable for a hardware implementation create the necessity of an inverse square root hardware. To perform this, a polynomial approximation is used, as presented in (SALMELA, 2008).

The original function to be approximated, $1/\sqrt{x}$, is highly non-linear. To make the approximation easier, the more linear $1/\sqrt{1+u}$ is used instead. The new input u is obtained by shifting the input x by α bits until it is in the format 1.u. If $x \geq 2$, then it will have to be shifted right to fit the desired format, and α will be negative.

$$x = (1+u).2^{-\alpha} \quad (5.4)$$

Using this definition, the function to be calculated can be modified:

$$\frac{1}{\sqrt{x}} = \frac{1}{\sqrt{(1+u).2^{-\alpha}}} = 2^{\alpha/2}\frac{1}{\sqrt{1+u}} \tag{5.5}$$

The multiplication by $2^{\alpha/2}$ can be performed by shifts only if α is even. For this reason, the approximation can be calculated by two different functions, thus allowing only shifters to be used.

For even values of alpha, k is defined as α/2:

$$\frac{1}{\sqrt{x}} = 2^{\alpha/2}\frac{1}{\sqrt{1+u}} = 2^{k}\frac{1}{\sqrt{1+u}} \tag{5.6}$$

For odd values of alpha, k is defined as (α-1)/2. The function to be calculated then becomes:

$$\frac{1}{\sqrt{x}} = 2^{\alpha/2}\frac{1}{\sqrt{1+u}} = 2^{\frac{\alpha-1}{2}}\sqrt{2}\frac{1}{\sqrt{1+u}} = 2^{k}\sqrt{2}\frac{1}{\sqrt{1+u}} \tag{5.7}$$

The functions to be approximated can use first order polynomials, due to the reduced non-linearity. The chosen polynomials' coefficients can be described as shifts. This eliminates the necessity for multipliers.

$$\frac{1}{\sqrt{1+u}} \cong 0.96582 - \frac{1}{4}u - \frac{1}{32}u \tag{5.8}$$

$$\sqrt{2}\frac{1}{\sqrt{1+u}} \cong 1.385742 - \frac{1}{2}u + \frac{1}{16}u \tag{5.9}$$

The resultant hardware structure is shown in Figure 5.6.

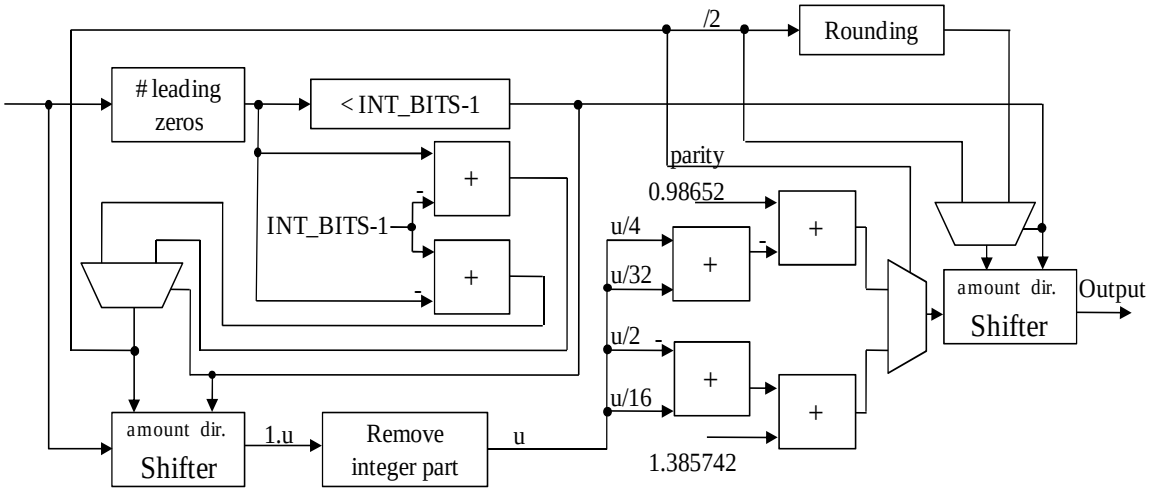

Figure 5.6: Inverse square root approximation hardware structure

Since 1 must be subtracted from α only when it is odd, the calculation of k can be simplified to a simple one bit right shift. However, as α is treated in the hardware by its modulus, accompanied by a shift direction bit, 1 must be added to k in the particular case of the shift direction being right and α being odd. This is due to the fact that the subtraction of one from α in the exponent would increase its modulus value, increasing the amount of bits to be shifted right. This operation is implemented by rounding the division by two in the shift right case.

The output of the block in Figure 5.6 is a first rough estimation of the desired value for the inverse square root. This approximation is not sufficiently precise to achieve a reasonably small increase in the FER, when compared to the floating point version of the QR decomposition, as can be seen in Figure 4.2. To improve this approximation, one Newton-Raphson iteration is used, with the equation deduced in section 3.2. Also, this block needs to be tested for possible overflows, which may occur when the input is too small. The minimum input I required is that which will produce the largest positive number L available in the representation as output:

$$\frac{1}{\sqrt{I}} = L \tag{5.10}$$

$$I = \frac{1}{L^2} \tag{5.11}$$

The complete inverse square root block is shown in Figure 5.7.

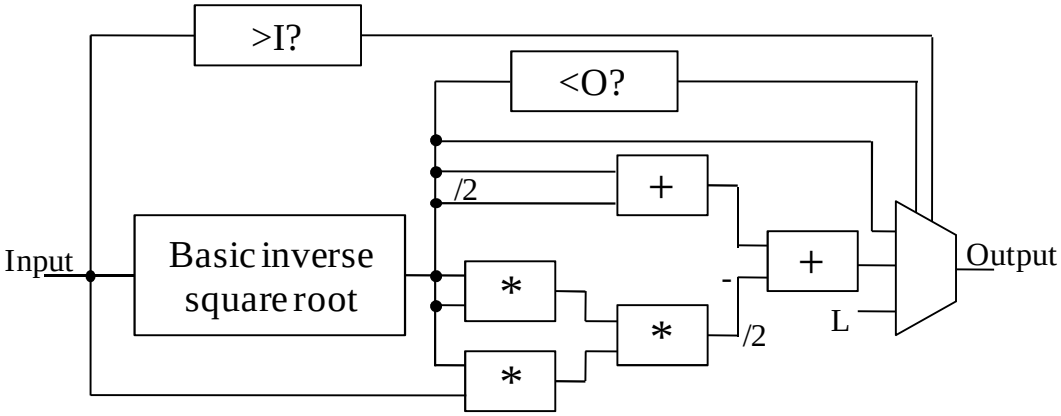

Figure 5.7: Inverse square root with improved approximation and overflow check

The original equation for the improvement of the result (equation 3.15) was slightly modified to make it more suitable for a hardware implementation:

$$y_{i+1} = y_i + \frac{y_i}{2} - \frac{y_i^2 . y_i x}{2} \tag{5.12}$$

When the output is too large, the multiplier that calculates its square will overflow. Even if saturation is used, the fine adjustment of the Newton-Raphson iteration is better left out, since the loss in precision caused by the saturation would actually make the result worse. For this reason, the application of the improvement is conditional, determined by a maximum value O for the basic block's output:

$$O^2 = L \tag{5.13}$$

$$O = \sqrt{L} \tag{5.14}$$

This way the output of the inverse square root is divided into three intervals:

- If the input is too small, the largest positive possible is the output;

- If the output of the basic inverse square root block is too large to be squared, the Newton-Raphson method is bypassed;

- Otherwise the output used is that of the Newton-Raphson approximation improvement hardware.

5.1.5 Vector operations

It is required to have a block for multiplying a vector by a scalar and a block for subtracting one vector from another. Both blocks can perform these operations in parallel or sequentially, using the basic complex multiplier and complex subtracter, with the trade-off between area and speed.

5.1.6 Matrices storage

The input matrix and output matrices need to be stored efficiently for computations. With the alterations in the MGS algorithm, the input matrix **H** and the output matrix **Q** can be stored in the same registers, with **Q** overwriting **H** as the algorithm proceeds. The **R** matrix, however, still needs its own storage area.

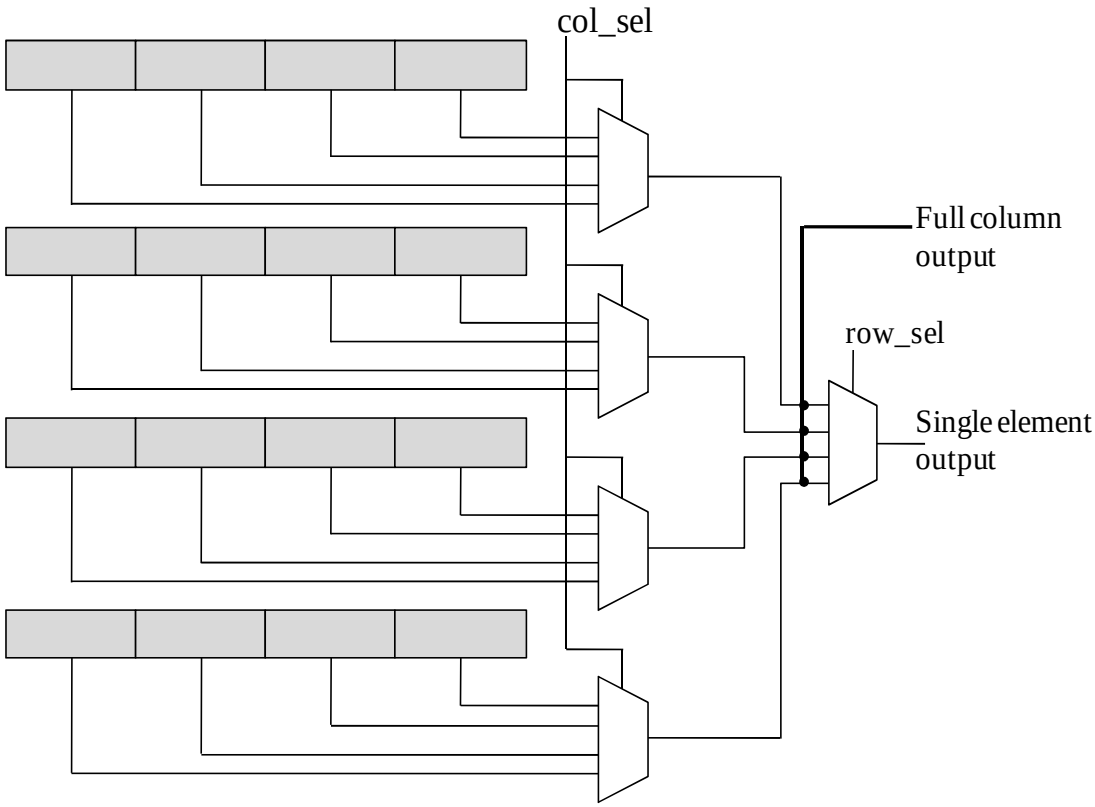

Figure 5.8: Basic matrix storage block

Figure 5.8 shows a simple way to organize the registers to store the matrices. By grouping the elements by line, it is possible to easily create an output for a whole column, which is very useful, as many computations are done over columns. The single element output, however, may still be necessary if sequential blocks are used, such as the sequential versions of the inner product or vector operations.

The writing lines are omitted in Figure 5.8. Unless an entirely sequential hardware is used, writings are going to be by column in **Q** and by single element in **R**.

Evidently, the storage block for **R** does not require registers for the elements under the main diagonal, as they are always zero.

5.1.7 Top level architecture

The basic blocks presented allow the creation of a top level architecture that interconnects them to compute the unsorted QR decomposition algorithm. Many

different combinations of sequential and parallel blocks are possible, specifically regarding the inner product and the vector operations.

By analyzing the algorithm, it is possible to determine how many times each block is used. For the 4×4 antennas example, the external loop is executed 4 times and the internal 7. This means that both the inner product and the vector multiplier are used 11 times, while the vector subtracter is used only 7. The first proposed architecture makes a compromise in the selection of the type of each block: the inner product is fully sequential and pipelined and the vector multiplier and subtracter are parallel. This is an attempt to create a design that is both sufficiently fast and with a reasonable area.

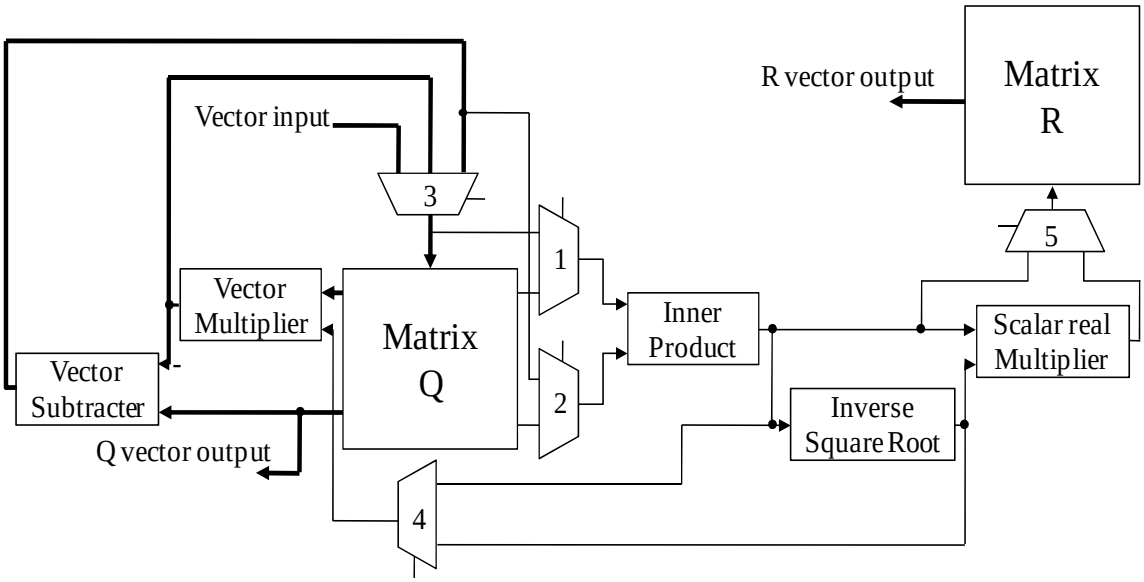

Figure 5.9: Top level architecture for unsorted QR decomposition

Figure 5.9 shows the complete top level architecture. The thick lines represent columns; the thin ones are for scalar elements.

To increase the parallelism, the storage element for the **Q** matrix has two full column and two single element outputs. However, the full column outputs actually share multiplexers with the single element ones, as in Figure 5.8.

Multiplexers 1 and 2 are for bypassing. Whenever the execution is entering the internal loop, the k^{th} column of **Q** is going to be used in the first input of the inner product block. However, this same column is being written, as it was just calculated in the vector multiplier block. The bypass multiplexer 1 allows taking the first element of the vector directly from the input of the matrix storage block, which, in this case, comes from the vector multiplier. The other situation in which bypassing is needed is when the last iteration of the external loop is beginning. In the last iteration of the inner loop, the last column of **Q** is written, and it is going to be used in the inner product of the external loop. Both inputs must come from the output of the vector subtracter, in this case. For both cases, only the first element of the vector is bypassed and all the others come from the memory block, as it will already have written the vector correctly by then.

5.1.8 Finite state machine

To control all the operations required to compute the QR decomposition, a finite state machine was obtained directly from the algorithm.

In Figure 5.10, a simplified version of this FSM can be seen, showing the main steps of the algorithm. During the *idle* state, the column selection is controlled by the external ports. This way, the output of the last execution can be read and the input for the next one can be written. The external port *start* triggers the execution of the algorithm with the matrix currently in the **Q** matrix registers. *Inner product 1* calculates the squared norm of the current column of **Q** and *Inv. Sqrt*. yields the inverse square root of the norm. During the *Mult* state, the computation of the current element of the main diagonal of **R**, which is the input of the inverse square root block multiplied by its output, is done in parallel with the calculation of the current column of **Q**, which is the product of itself with the output of the inverse square root. The transition to the internal loop is done with the first bypass activated (multiplexer 1 in Figure 5.9). *Inner product 2* calculates the elements of **R** that are not in the main diagonal and the *Mult and sub* state performs the subtraction of the projection of $\mathbf{q}_j$ on $\mathbf{q}_k$. The return to the *Inner prod 1* state is done with both bypasses activated, but only if the last iteration of the external loop is beginning. The computation is finished in the *Mult* state, since the internal loop is not executed in the last iteration of the external one.

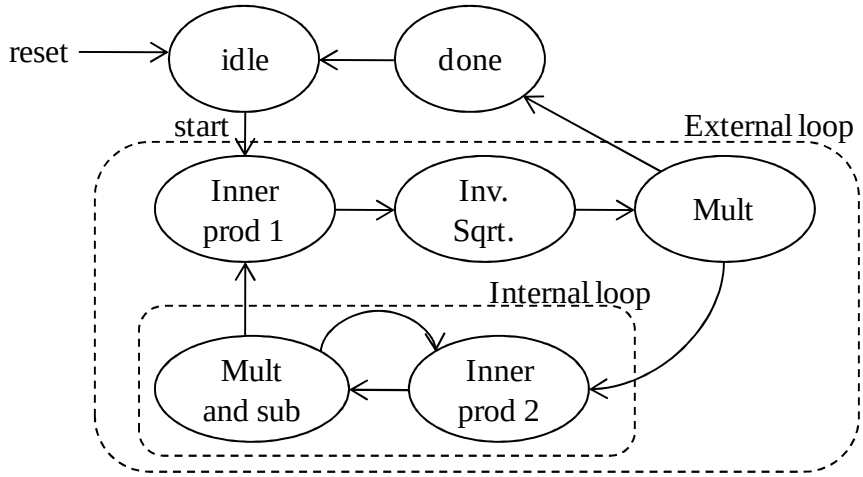

Figure 5.10: Simplified FSM for the unsorted QR decomposition hardware

5.2 Sorted QR Decomposition

Most of the basic building blocks for the SQRD hardware are the same as for the unsorted version, with the exception of the matrix storage elements. The main changes in the data path and in the FSM are also presented in the following sections.

5.2.1 Matrix storage elements

The storage elements need to be modified in order to allow efficient swapping operations. These operations are performed internally in each storage element, due to performance issues. The same signals that are connected to the vector outputs (for the case of the **Q** matrix) can be used as inputs for the registers, and all swapping operations can be performed in a single cycle, greatly reducing the added amount of time. For the **R** matrix, extra column selection multiplexers were added to allow this operation, creating a second column output that is accessible only internally. Figure 5.11 shows the basic swapping schematic for one matrix row.

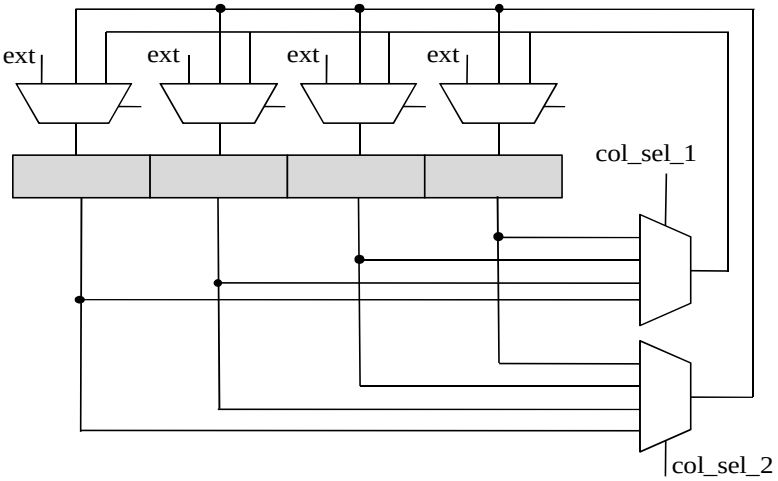

Figure 5.11: Swapping hardware for one row

5.2.2 Top level architecture

The main changes in the data path structure are the introduction of the norm and permutation vectors and the norm update hardware. Both vectors are stored in elements similar to the ones used for the matrices' lines, as presented in Figure 5.11, to allow fast swapping.

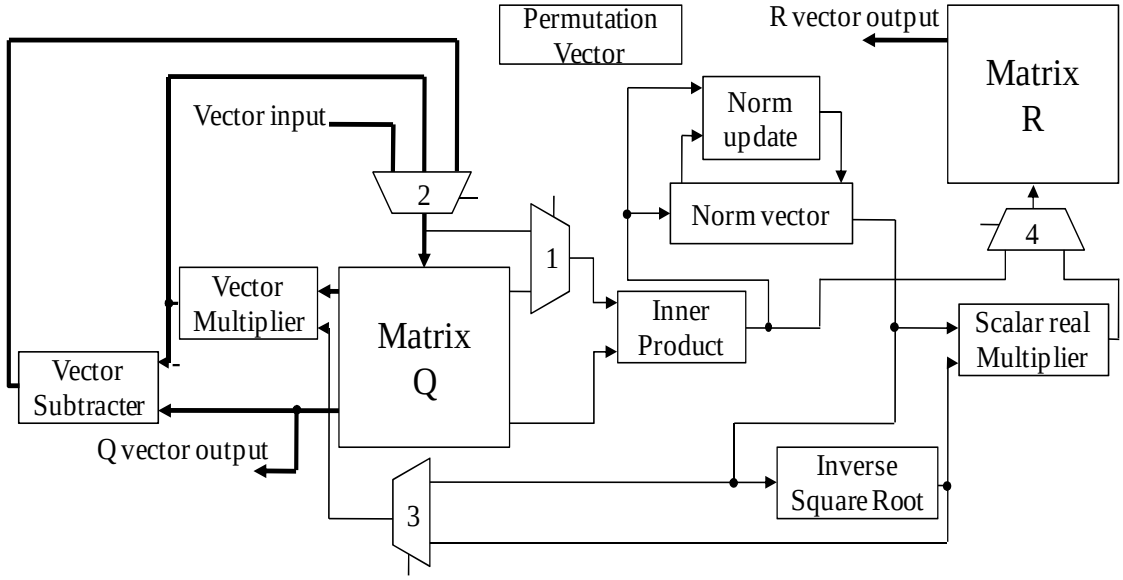

Figure 5.12: Top level architecture for sorted QR decomposition

Figure 5.12 presents the resulting architecture. Since all dot products are calculated before the beginning of the external loop, the bypass multiplexer for the second port of the inner product is no longer necessary (multiplexer 2 in Figure 5.9).

The permutation vector is not connected to any other block, since it only records the swaps that are done in the matrices and in the norm vector.

5.2.3 Finite state machine for SQRD

Figure 5.13 shows the FSM to perform the SQRD. It is derived from the one used for unsorted version of the hardware and the alterations that led to Algorithm 5.

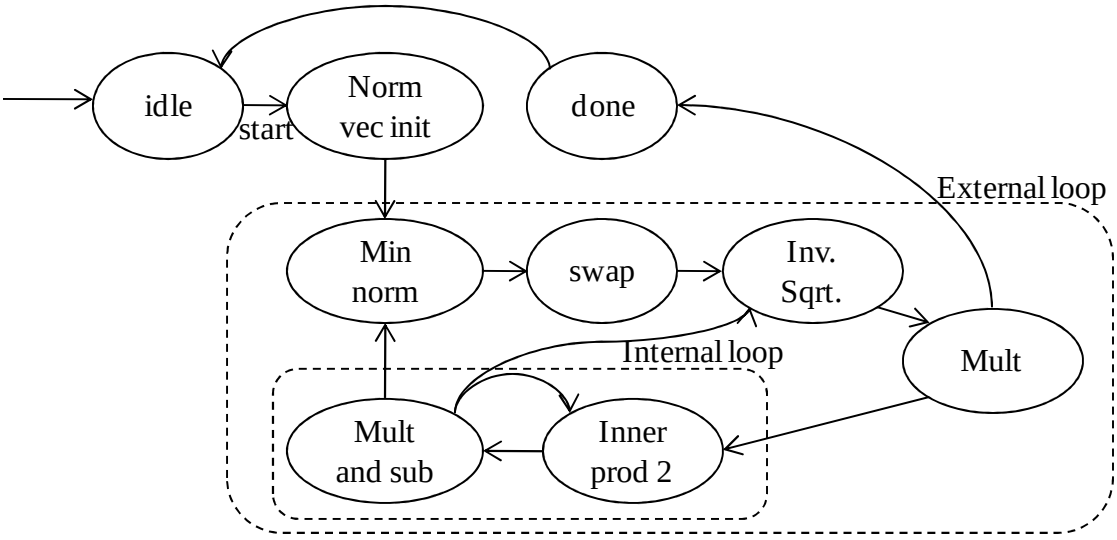

Figure 5.13: Simplified FSM to perform the SQRD algorithm

The states that were added are *Norm vec init*, *Min norm* and *swap*. The first one initializes the norm vector with the squared norm of each column of the input matrix. The second one searches for the column with the smallest norm among the ones that are still left to be calculated and the *swap* state activates the signals to perform the required swap operations in the **Q** and **R** matrix, as well as in the norm and permutation vectors.

The norm update is performed in parallel with the multiplication and subtraction already present in the internal loop (*Mult and sub* state). Once it is finished, it can return to the *Min norm* state or skip directly to the *Inv. Sqrt.* state, since in the last iteration no minimum norm search or swap operations are required.

5.3 MMSE Sorted QR Decomposition

As occurred for the original SQRD algorithm, most of the basic blocks necessary for the MMSE-SQRD are already defined. The finite state machine is also basically the same used for the SQRD hardware, since the algorithm itself is almost the same. The main challenge here is to develop a hardware that is not excessively larger and not excessively slower, when compared to the other versions, since we are now working with vectors that have 2N elements. And it is also necessary to create an efficient way to store the modified **Q** matrix, which has several different properties when compared to the previous versions of the algorithm.

5.3.1 Storage element for the modified Q matrix

When using the MMSE-SQRD pre-processing, the **Q** matrix is extended. More specifically, it becomes a 2N×N matrix. Also, its top ($\mathbf{Q_1}$) and bottom ($\mathbf{Q_2}$) parts have several different properties. While $\mathbf{Q_1}$ is initialized with the input matrix, $\mathbf{Q_2}$ is initialized with $\sigma\mathbf{I}_N$. Also, $\mathbf{Q_2}$ is upper triangular, since it is the scaled inverse of **R**. Another difference is that, when a swapping operation is done, not all rows of $\mathbf{Q_2}$ are swapped, but only the first i. For all these reasons, and also to allow a larger parallelism in the required operations, the $\mathbf{Q_1}$ and $\mathbf{Q_2}$ matrices are stored in separate blocks.

By storing these two matrices in two different blocks, one can easily explore the main differences of the matrices for better results. The $\mathbf{Q_1}$ matrix is actually identical to the **Q** matrix of the SQRD algorithm, since it is also initialized with the input matrices and, when a swap operation is done, all of its rows are exchanged. The $\mathbf{Q_2}$ matrix, on the

other hand, does not need registers for the elements under the main diagonal, as the **R** matrix. The swap automatically exchanges only the required columns and the reset port loads the initial matrix in one single cycle.

5.3.2 Top level architecture

As mentioned, the MMSE-SQRD architecture needs to deal with the extended vectors in a way that does not increase excessively the area nor the amount of cycles required.

In order not to introduce too many extra cycles, one extra inner product block is used. Since these blocks are sequential, using only one complex multiplier each, the extra area is not too expensive. Each block operates on one half of the **Q** matrix, and in the end the results are added, yielding the desired dot product. The vector operations, namely multiplication and subtraction, however, are done in parallel, which means that adding one extra block of each can be overly expensive. Pipelining these operations, on the other hand, adds little extra area and only one extra cycle. Therefore, both the vector multiplier and subtracter remained with the same size of the previous versions, i.e. N elements in parallel, and operate in pipeline, first processing a column of $\mathbf{Q_1}$, then a column of $\mathbf{Q_2}$. Figure 5.14 shows the resultant hardware schematic.

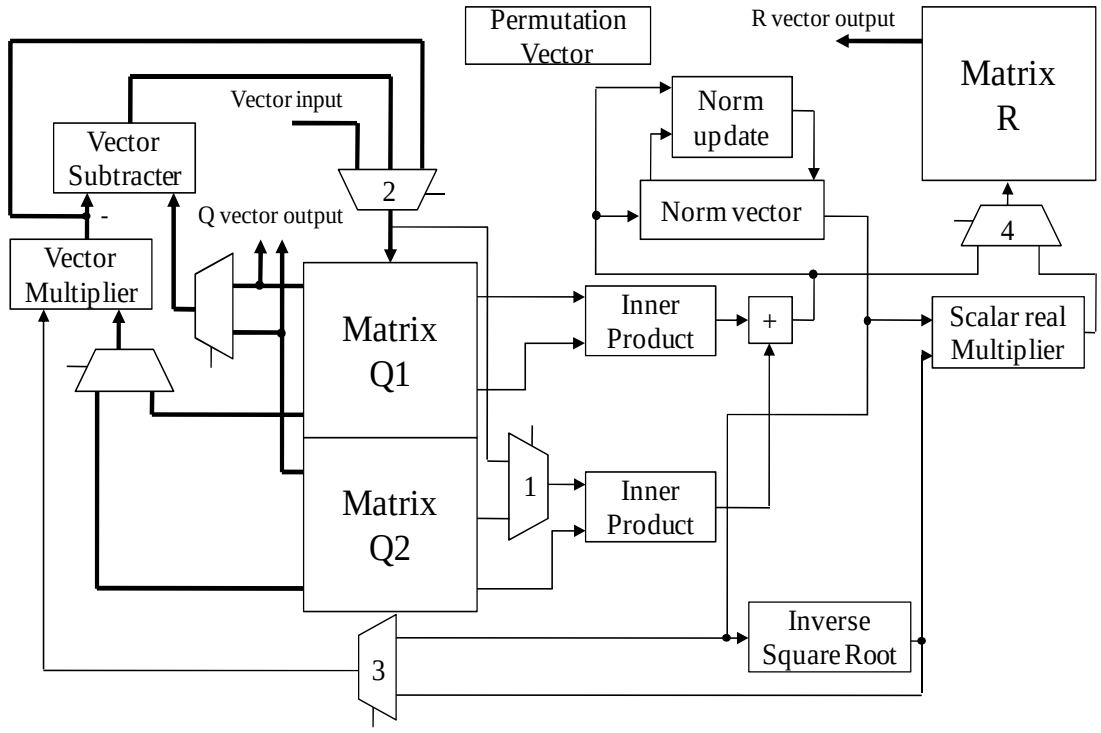

Figure 5.14: Top level architecture for the MMSE-SQRD algorithm

Note that the bypass multiplexer 1 now is required only in the input of the second inner product block, since the vector operations are done in pipeline, and the columns of $\mathbf{Q_1}$ have one extra cycle to be written, therefore being available at the input of the inner product when required.

5.4 Reduced order matrices

One extra requirement for all QR decomposition hardware implementations is that they must work with two different matrix sizes. Aside from the maximum capacity, when the hardware is used to its maximum, they must be able to work with matrices that

have their order divided by two. For example, the hardware synthesized to work with 4×4 matrices must be able to work with 2×2 matrices as well.

This could be done, for the unsorted case, directly by placing the smaller matrix on the top left corner of the larger input matrix and leaving the rest of the input with zeroes, with the hardware completely unaware of the reduced order. The desired **Q** and **R** matrices would come out automatically on the top left corner.

However, this would mean that many cycles are wasted, since the decomposition time is highly dependent on the matrices' order. Also, the sorted versions would not work, because the zeroed columns would interfere with the sorting. For these reasons, one external control port must be added to the designs. When activated, this port modifies all FSM cycle counts that are related to the matrices' orders, which include the external and internal loop iterations and also the amount of inputs expected by sequential operators, more specifically, the inner product blocks, thus greatly reducing the total execution time.

6 HARDWARE IMPLEMENTATION RESULTS

The three main variations of the QR decomposition algorithm, i.e. unsorted QRD, SQRD and MMSE-SQRD, were implemented in VHDL, following the architectures described in chapter 5. These implementations were synthesized and had their results analysed for Xilinx FPGAs of the Virtex-5 family and also using Synopsys Design Compiler with STMicroelectronics 65nm library.

6.1 Timing results

All hardware variations were designed to achieve a clock frequency of 200MHz in Xilinx Virtex-5 FPGAs. These same designs reached up to 500MHz when implemented for ASIC, using the mentioned library. In order to achieve these clock frequencies, some timing barriers were added to critical combinational paths. Also, a multi-cycle path was defined for the inverse square root hardware, which takes 4 four cycles to finish its computation in FPGA and 3 cycles in ASIC, including the Newton-Raphson approximation improvement. A multi-cycle path is a way to define that the combinational path between two timing barriers does not need to be executed in one single cycle. This way, the tool can correctly calculate the resulting clock frequency and also spend the appropriate optimization effort in that path. It is still, however, required that the designer ensures that the correct amount of cycles is given to the hardware to perform its execution.

Aside from the clock frequency, the amount of cycles spent for each QR decomposition is important to determine the total computation time. This number is evidently different for each algorithm and for each matrix size. Figure 6.1 shows the results for a 4×4 matrices hardware, also working with 2×2 matrices, when the reduced order external port is activated.

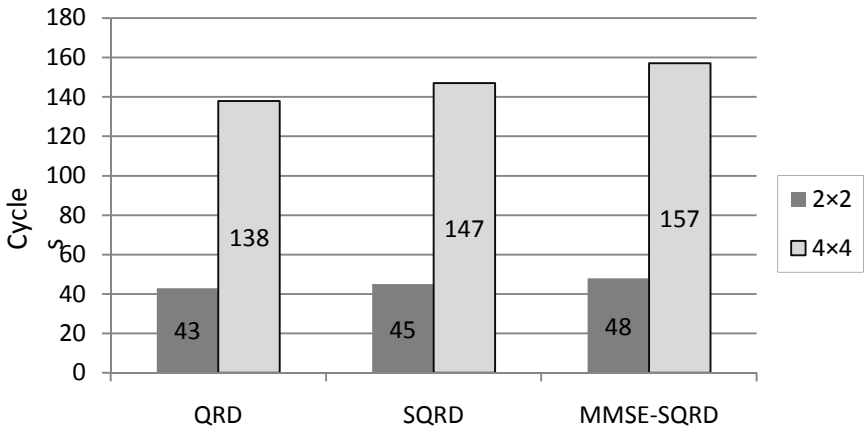

Figure 6.1: Amount of cycles for each QR decomposition version

The results in Figure 6.1 consider 3 cycles for each inverse square root execution, i.e. the time taken by the ASIC implementation. The total execution time for FPGA versions is 4 cycles longer for 4×4 and 2 cycles longer for 2×2.

6.1.1 Real time requirements

The real-time requirements are related to the target communication technology and target speed of the receiver. They are calculated using the coherence time t_{coh}, which is the time in which the channel impulse response is essentially invariant (SALMELA, 2008). It is given by:

$$t_{coh} = \frac{c}{v_r f} \tag{6.1}$$

Where c is the speed of light (3×10^8m/s), v_r is the receiver speed and f is the carrier frequency. Considering, for example, 3G LTE MIMO systems, f is 2.4GHz. Considering a maximum receiver speed of 250km/h, t_{coh} = 1.8ms. However, bullet train speeds can also be considered (SALMELA, 2009), such as 500km/h. This results in t_{coh} = 0.9ms. During this time, the QR decomposition for all the 1021 sub-carriers of the OFDM modulation proposed for such systems must be computed (BACHL, 2007). This leaves us with 1,763μs for each decomposition, considering the first constraint, and 0,881μs, considering the second one.

The FPGA implementation takes 161 cycles to compute one 4×4 MMSE-SQRD, which is the one that takes the longest. Since its maximum frequency is 200MHz, it takes 0,805μs to perform this computation. Therefore, it is capable of meeting even the strictest constraint. Since the ASIC implementation takes less cycles, it can obviously also satisfy these constraints running at 200MHz. Future research, however, may require faster decompositions, as can systems that work with higher carrier frequencies. For these reasons, the higher operation frequencies of the ASIC implementation are also analysed in sections 6.2 and 6.3.

6.2 Area results

The different algorithms to perform the QR decomposition had their area results analysed also with FPGA and ASIC synthesis tools. The results between each different algorithm and different bit widths are presented and compared in the following sections. Also, for the ASIC implementations, different target clocks are compared.

6.2.1 FPGA area results

Using Xilinx ISE to synthesize the different hardware variations for a Virtex-5 XC5VFX30T FPGA, the total amount of used registers, LUTs and multipliers can be compared between each implementation.

Virtex-5 FPGAs use blocks called DSP48E for multiplication. Each block contains one 25×18 bits multiplier, one adder and one accumulator. Since all bit widths used are under a total of 18 bits, the amount of DSP48E blocks used is not bit width dependant.

6.2.1.1 Comparison between different algorithm versions

The three different algorithm variations are compared using the specific amount of bits required to not introduce significant degradation in the frame error rate. The original QRD uses the 4.8 format, determined in chapter 4. Figure 6.2 shows the results in LUTs and registers. Since the SQRD algorithm implemented uses the norm update

optimization, i.e. it initializes the norm vector at the beginning of execution and then only updates it, it requires 4 integer and 9 fractional bits, as shown in Figure 4.11. Even though the MMSE-SQRD version also uses the norm vector, no significant FER increase was observed when using a 4.8 format (Figure 4.21). For this reason, this is the chosen format for MMSE-SQRD versions.

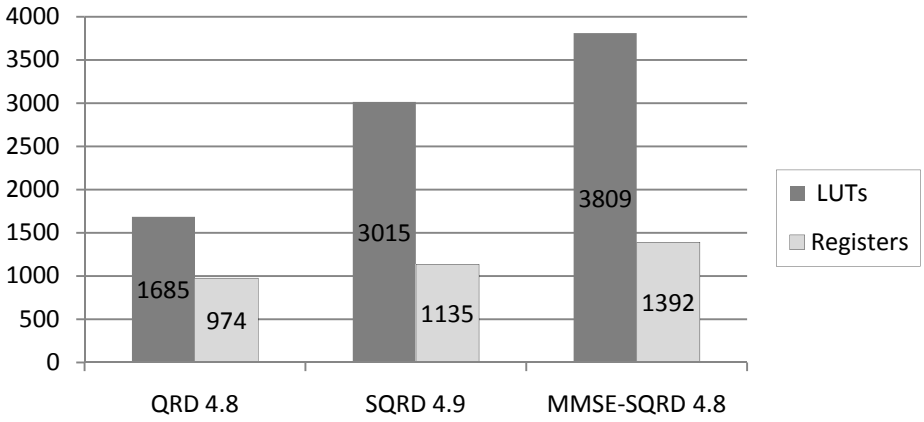

Figure 6.2: Slice LUTs and registers for different algorithms in FPGA

The amount of used DSP48E blocks also varies between each implementation. The QRD hardware uses a total of 24 multipliers: 16 in the vector multiplier (4 complex multipliers with 4 multipliers each); 4 in the inner product block (one complex multiplier); 3 in the inverse square root block (for the Newton-Raphson method) and 1 to calculate the elements in the main diagonal of **R**. The SQRD version uses 26 multipliers, with the two extra ones being used to calculate the norm update, which is done using half complex multiplier, since the imaginary part of the result is always zero. Finally, the MMSE-SQRD uses 30 multipliers, where the extra 4 ones are used in the second inner product block.

6.2.1.2 Comparison between different bit widths

The original QRD algorithm was synthesized for different number formats to evaluate the area progression as the precision is increased. Figure 6.3 shows this progression for bit widths ranging from 11 to 14, always with 4 integer bits.

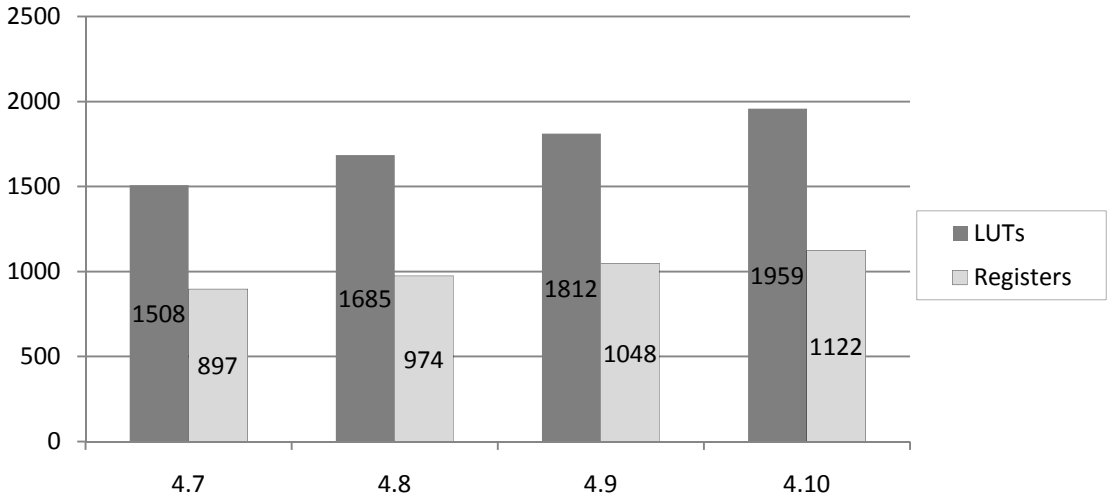

Figure 6.3: FPGA area results for QRD using different number formats

6.2.2 ASIC area results

As was done for FPGA, the area results can be compared for different algorithm versions and different bit widths for ASIC implementations. Also, the Synopsys Design Compiler works with a target clock frequency, which has significant impact on the resultant area. Therefore, the area progression is also evaluated for different target clock frequencies. All results are using STMicroelectronics 65nm library and in μm^2, the default output unit for this library.

6.2.2.1 Comparison between different algorithm versions and clock frequencies

The three different algorithms were synthesized for different clock frequencies, ranging from 200MHz to 500MHz. The main difference between these results and the ones obtained for FPGAs is that these include all area in a single value, i.e. combinational logic, memory units and multipliers, which are obtained as separated values when using FPGAs. Again, the only algorithm using the 4.9 format is the SQRD.

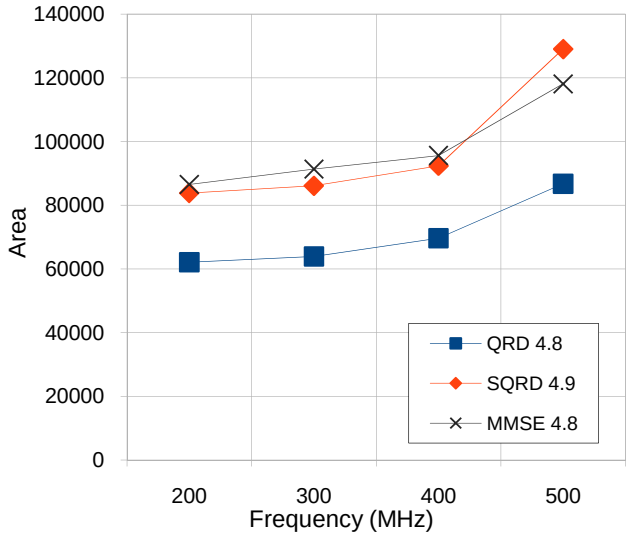

Figure 6.4: ASIC area results for different algorithm versions and clock frequencies

Figure 6.4 shows that the ratio between the area for the QRD and the MMSE-SQRD versions is much more favourable for the latter when implemented for ASIC then when for FPGA. While the amount of LUTs used by MMSE is more than two times that which was used by the original QRD, here the area is only around 39% larger for 200MHz and 37% for 500MHz, for example. Aside from different tool optimization algorithms, which may have better results for one algorithm than for the other, this can be explained by the required amount of registers and multipliers. These values are counted separately for FPGAs, and have ratios more favourable for the MMSE version then that of the required amount of LUTs.

Also, it can be seen that, when working with 500MHz, the area for MMSE-SQRD is actually smaller than that for SQRD. This due the extra bit required by SQRD, which has a more significant impact in the area when we approach the clock frequency limit, since the tool will need a larger effort to meet this constraint in all paths.

6.2.2.2 Comparison between different bit widths

The same bit widths used for FPGA analysis (from 11 to 14) were synthesized using Synopsis DC, also using 4 integer bits. Figure 6.5 shows the results for the original QRD algorithm and frequencies ranging from 200MHz to 500MHz.

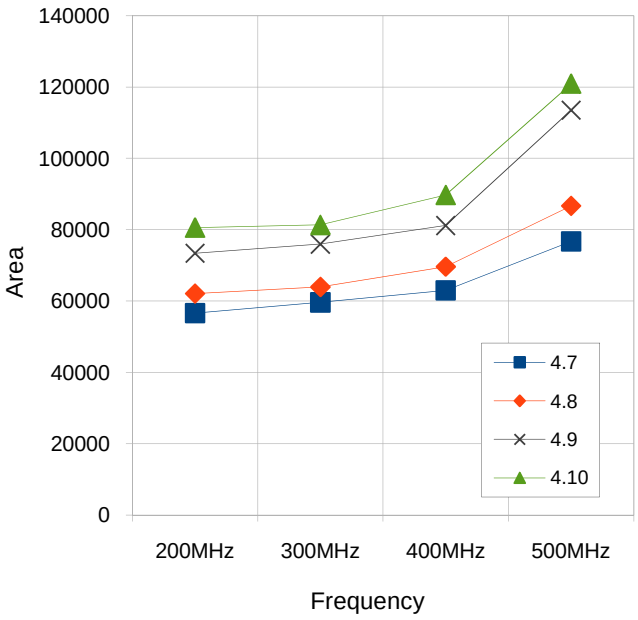

Figure 6.5: ASIC area results for QRD using different number formats and clock frequencies

The result for the 4.10 format and 500MHz is actually for a frequency of 497.51MHz, which is the maximum frequency that could be reached using this bit width.

Again, it is visible that largest area increases are observed when we approach the maximum frequency, especially when dealing with larger bit widths.

6.3 Comparison with high-level synthesis

For further evaluation of the obtained results, one can compare them with those of an automatic high-level synthesis tool. Here, Mentor Graphics' Catapult was used. This tool transforms pure ANSI C++ code directly into VHDL and Verilog RTL descriptions. It also identifies loops and allows the user to determine which ones will be left rolled and which ones will be unrolled and parallelized.

Using Catapult's unrolling functionality, the QR decomposition C code was synthesized in the most similar way to that which was determined for the manual VHDL implementation: vector operations (multiplications and subtraction) were unrolled and inner products and the main loops (both internal and external) were left rolled. The ac_fixed and ac_complex types, the same ones used in the simulation chain, allow the definition of fixed point formats and usage of rounding and saturation. All results in this section are for the unsorted QRD with a 4.8 fixed point format.

Aside from the RTL descriptions, Catapult also outputs latency and area results. These results are based on a sample library chosen before synthesis, which is the same one used by the tool to determine which operations can be executed in a clock period of the target frequency.

The following graphics compare the results obtained with Catapult's 65nm sample library and the ones from section *6.2.2.1*, obtained with Synopsys DC and STMicroelectronics 65nm library. Also, for comparison purposes, the latency-area product is plotted. This product allows a quick comparison of the quality of two

solutions, since latency versus area trade-offs such as the choice between parallel and sequential hardware have little impact on it.

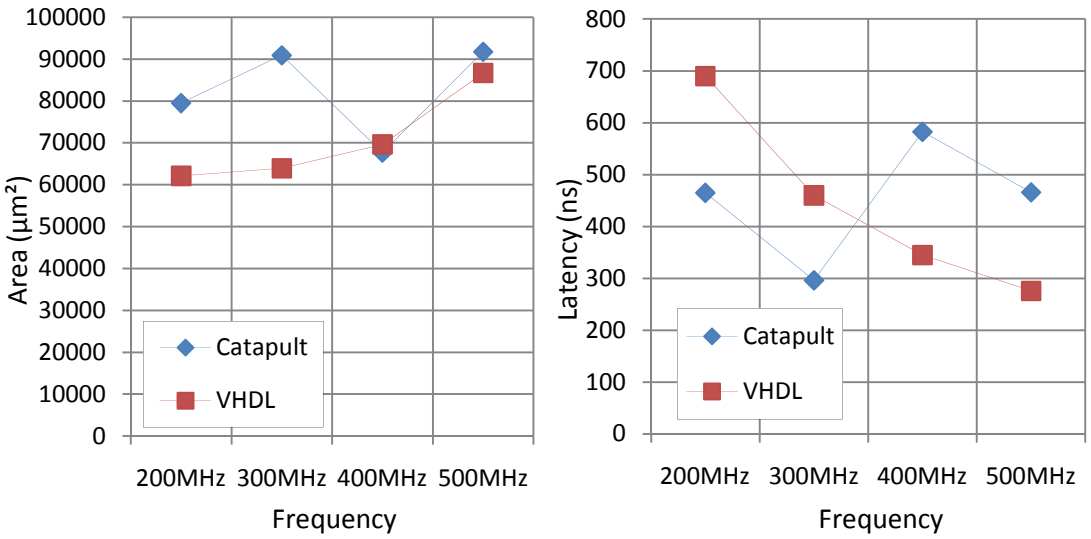

Figure 6.6: Comparison between Catapult and manual VHDL implementations

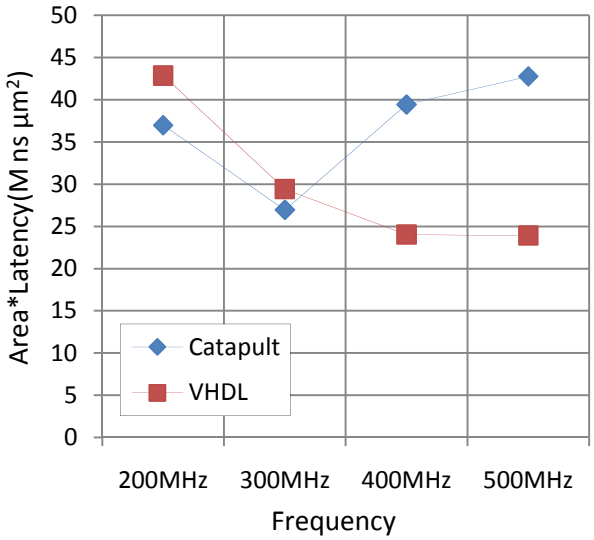

Figure 6.7: Latency-area product for Catapult and manual VHDL implementations, in millions of ns.µm^2

Figure 6.6 shows that the area is smaller for the manual implementation for all frequencies except 400MHz, for which Catapult's version is slightly smaller. Catapult's versions are faster than the manual ones for 200MHz and 300MHz. This is mainly due to the fact that all manual implementations use the same timing barriers, which were positioned for a high frequency design, and therefore take the same amount of cycles. The high-level tool can reposition all barriers optimizing for the current target frequency, thus greatly reducing the amount of required cycles for lower frequencies. Another interesting fact is that increasing the target frequency in Catapult does not necessarily reduce the delay.

The latency-area graphic, in Figure 6.7, shows that Catapult achieves its optimal result for a target frequency of 300MHz, while the manual versions best results are for 400MHz and 500MHz, which yield the best latency-area products among all variations.

It is important to emphasize, though, that these results use different libraries, which may have different area and latency values, even though they both use 65nm technologies. Perhaps a more accurate and fair comparison is to take the VHDL generated by Catapult for each target frequency and obtain results from Synopsys DC with the exact same library as used for the manual versions.

When trying to do this, however, one important observation immediately comes to hand: RTL descriptions generated for a given target frequency by Catapult do not always have their timing constraints successfully met for the same frequency in Synopsys DC. This means that Catapult's sample library is probably more optimistic about delays than a real library. Most notably, the outstanding 300MHz version cannot be synthesized for 300MHz.

Using Catapult-generated designs in Synopsys DC, 200MHz were successfully reached with the code generated for this same frequency. Catapult designs targeting 300, 350 and 380MHz could not be successfully synthesized for 300MHz. A design targeting 400MHz had to be used, which takes 233 cycles instead of 89 for a single decomposition. Finally, 400MHz could not be reached, even when using designs that were synthesized for frequencies as high as 600MHz. Also, for both successful synthesis cases, the reported area was significantly larger than that informed by Catapult. The following graphics show the comparison between the results.

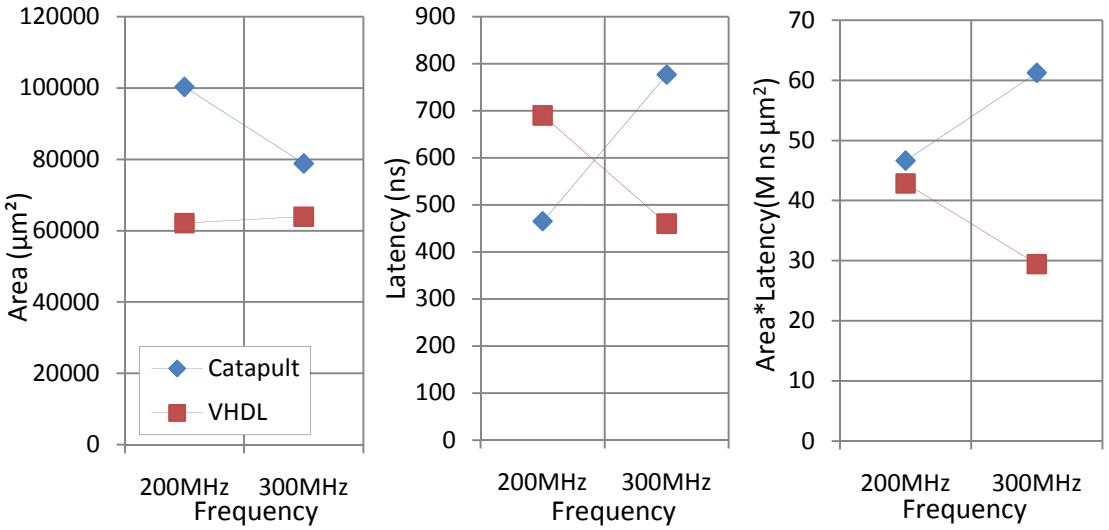

Figure 6.8: Area, latency and latency×area comparison between manual VHDL and Catapult versions synthesized with Synopsys DC

Figure 6.8 shows that Catapult's inability to maintain a low cycle count for higher frequencies and the large area increase for 200MHz placed both versions' latency-area products above those which were obtained for the manual VHDL implementations.

7 CONCLUSIONS

The specific requirements for the QR decomposition for sphere decoding were analysed and the chosen algorithm, the Gram-Schmidt process, received the appropriate modifications and improvements to make it more suitable for a hardware implementation. Also, two pre-processing improvements, namely the sorted QR decomposition and the minimum mean squared error SQRD, were defined and had their impacts analysed in the system. The analysis considered both error rates and computational effort, measured by the amount of visited nodes in the tree search of the sphere decoder. The simulations showed that both techniques have beneficial effects in the overall system, significantly reducing the amount of visited nodes without significant changes in the error rates. Average reductions of up to 74% could be reached, when coupling MMSE-SQRD and OSS, in a 4×4 system.

The three pre-processing variations had their hardware architectures specified, considering the required data path and finite state machine for each one. The proposed architectures were implemented in VHDL and synthesized for the required bit widths, and were able to reach relatively high frequencies both in FPGAs (200MHz) and in ASICs (500MHz). The amount of cycles necessary was also reasonable, when compared to other implementations, such as in (SALMELA, 2008). The increases when using SQRD and MMSE-SQRD are tolerable, since the system was able to meet a real-time constraint for a real projected technology, 3G LTE, even for the FPGA implementation of the MMSE-SQRD version, considering a receiver at 500km/h.

The area increase when comparing the different algorithms is most expensive when FPGA LUTs are taken for measure. Considering that other resources, more specifically registers and multipliers, are not increased by the same factor, the comparison becomes less unfavourable for the improved versions. This comparison is also less unfavourable for the larger versions when ASIC area is considered, especially for MMSE-SQRD. In this case, the average increase, when compared to the original QRD, is under 40% for the analysed clock frequencies.

REFERENCES

BACHL, R. *et al.* The long term evolution towards a new 3GPP* air interface standard. **Bell Labs Technical Journal**, v. 11, n. 4, p. 25–51, 2007.

FINCKE, U.; POHST, M. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. **Mathematics of Computation**, v. 44, n. 4, p. 463-471, 1985.

GILBERT, F.; KIENLE, F.; WEHN, N. Low Complexity Stopping Criteria for UMTS Turbo-Decoders. In: SPRING VEHICULAR TECHNOLOGY CONFERENCE, 2003, **Proceedings...** Jeju, Korea: IEEE, p. 2376–2380.

GIMMLER, C. **Performance and complexity of iterative sphere/channel decoding in MIMO systems**. 2007. 83 f. Thesis (Diploma Thesis) - Fachbereich Elektrotechnik und Informationstechnik, Technische Universität Kaiserslautern, Kaiserslautern.

GOLLUB, G. H.; VAN LOAN, C. F. **Matrix Computations.** 3rd ed. Baltimore, MD: John Hopkins University Press, 1996.

LUETHI, P. *et al.* Gram-Schmidt-Based QR Decomposition for MIMO Detection: VLSI Implementation and Comparison. In: ASIA PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS, 2008, **Proceedings…** Macao, China: IEEE, 2008, p. 830–833.

MENNENGA, B.; FRITZSCHE, R.; FETTWEIS, G. P. Iterative Soft-In Soft-Out Sphere Detection for MIMO Systems. In: VEHICULAR TECHNOLOGY CONFERENCE, 2009, **Proceedings...** Barcelona, Spain: IEEE, 2009.

PRESS, W. H. *et al.* **Numerical Recipes in C.** 2nd ed. Cambridge: Cambridge University Press, 1992.

SALMELA, P. e*t al.* 3G long term evolution baseband processing with application-specific processors. **International Journal of Digital Multimedia Broadcasting**, v. 2009, 2009.

SALMELA, P. *et al.* Complex-valued QR decomposition implementation for MIMO receivers. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 2008, **Proceedings…** Las Vegas, NV: IEEE, 2008, p. 1433-1436.

SIMHA S, V. **Fixed Point Implementation of Soft Input/Soft Output Sphere Decoder and EXIT Chart Analysis**. 2009. Thesis (Master Thesis) - Fachbereich Elektrotechnik und Informationstechnik, Technische Universität Kaiserslautern, Kaiserslautern.

STUDER, C.; BURG, A.; BÖLCSKEI, H. Soft-Output Sphere Decoding: Algorithms and VLSI Implementation. **IEEE Journal on Selected Areas in Communications**, v. 26, n. 2, p. 290–300, 2008.

TELATAR, I. E. Capacity of multi-antenna Gaussian channels. **European Transactions on Telecommunications**, v. 10, n. 6, p. 585–595, 1999.

WÜBBEN, D. *et al.* Efficient Algorithm for Decoding Layered Space-Time Codes. **IEE Electronics Letters**, v. 37, n. 22, p. 1348–1350, 2001.

WÜBBEN, D. *et al.* MMSE Extension of V-BLAST based on Sorted QR Decomposition. In: VEHICULAR TECHNOLOGY CONFERENCE, 2003, **Proceedings...** Orlando, FL: IEEE, 2003, p. 508-512.

APPENDIX

A.1 Additional simulations with sphere shrinking and SQRD

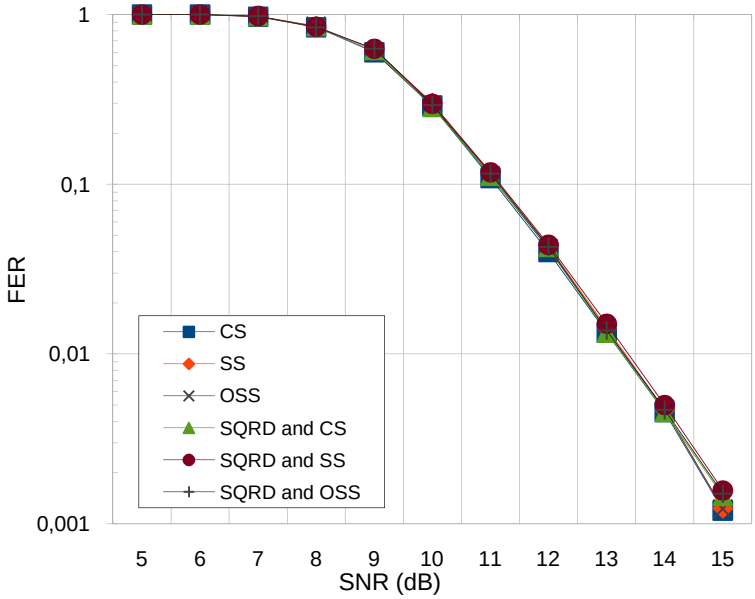

Figure A.1: FER for different algorithms with fixed point in a 4×4 antennas system

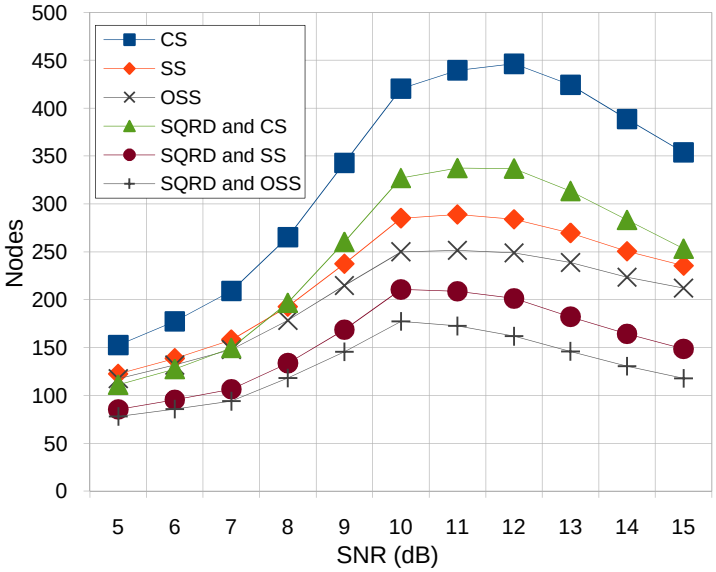

Figure A.2: Average amount of visited nodes with fixed point in a 4×4 antennas system

Table A.1: Parameters for simulations in Figures A.1 and A.2

Parameter	Value
Number of antennas	4
Frame errors limit	100
Number format	Fixed point 4.8

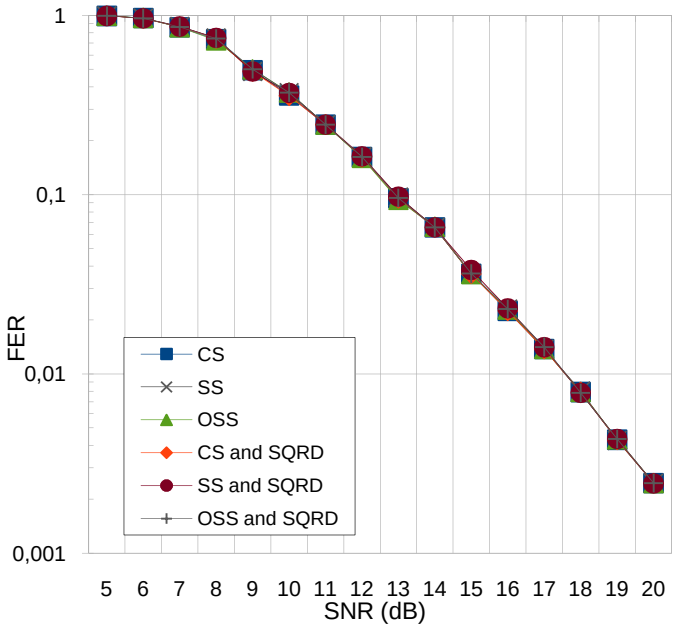

Figure A.3: FER for different algorithms with floating point in a 2×2 antennas system

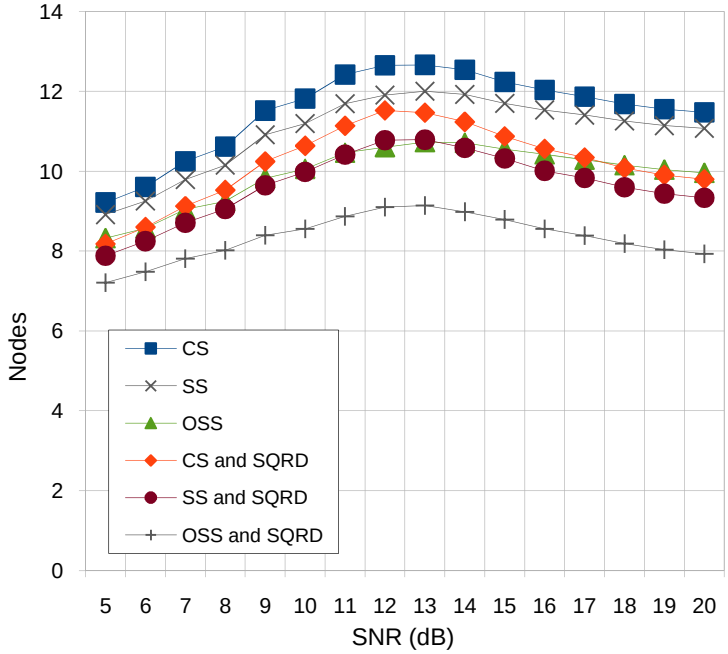

Figure A.4: Average amount of visited nodes with floating point in a 2×2 antennas system

Table A.2: Parameters for simulations in Figures A.3 and A.4

Parameter	Value
Number of antennas	2
Frame errors limit	150
Number format	Floating point

The results found for 2×2 antennas agree with the ones for 4×4. However, as the amount of intermediate nodes in the tree is much smaller, the effects of most algorithm combinations cannot be seen so clearly. Still, it is even clearer in this case that the combination of SQRD with OSS has outstanding performance.

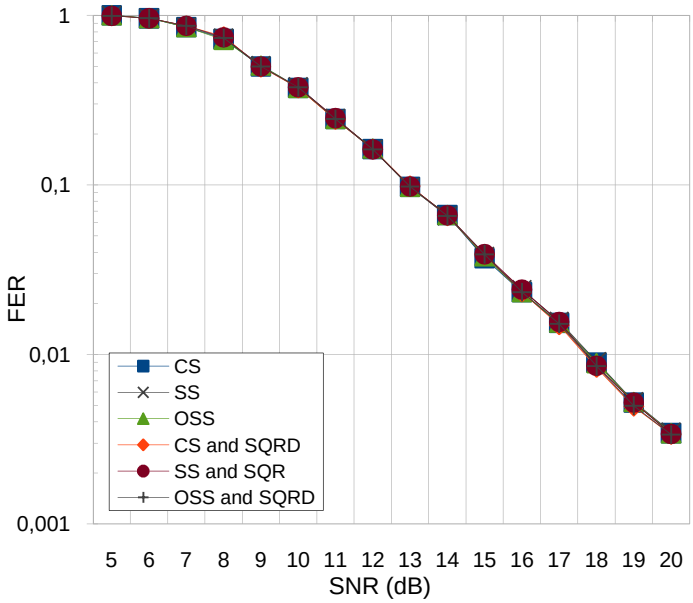

Figure A.5: FER for different algorithms with fixed point in a 2×2 antennas system

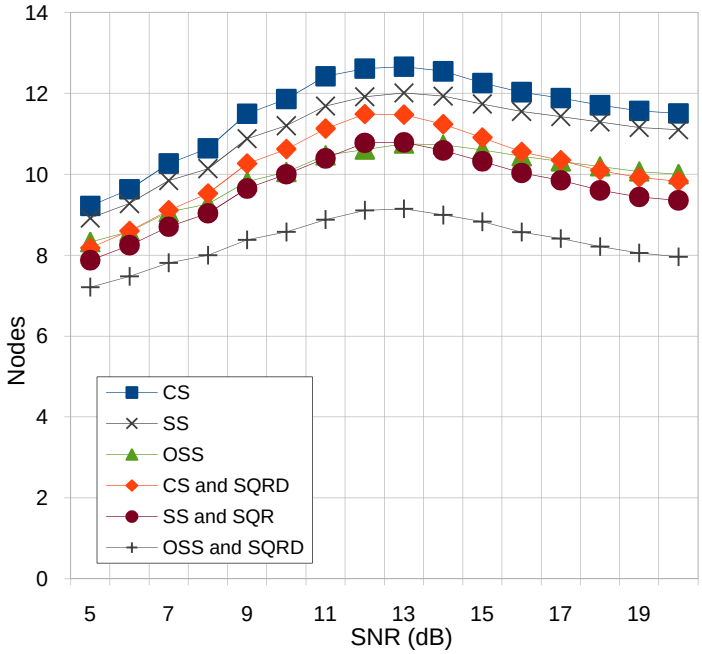

Figure A.6: Average amount of visited nodes with fixed point in a 2×2 antennas system

Table A.3: Parameters for simulations in Figures A.5 and A.6

Parameter	Value
Number of antennas	2
Frame errors limit	150
Number format	Fixed point 4.8

A.2 Simulation results with 64-QAM

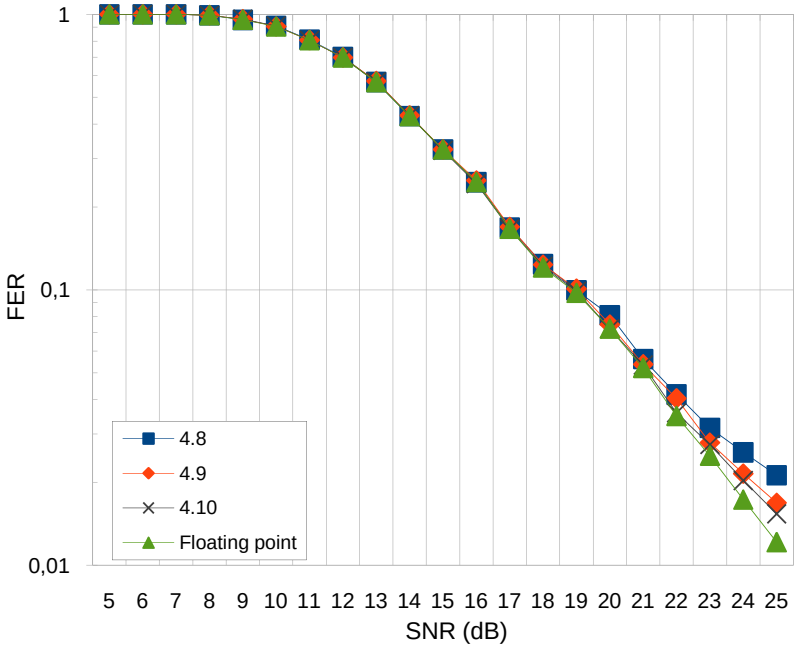

Figure A.7: Required amount of fractional bits for a 64-QAM 2×2 antennas system

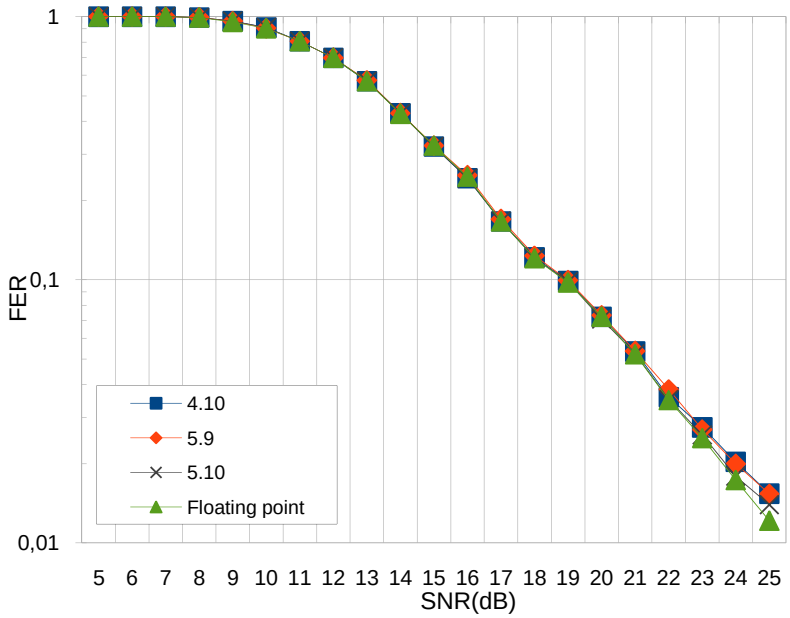

Figure A.8: Required amount of integer bits for a 64-QAM 2×2 antennas system

Table A.4: Parameters for simulations in Figures A.7 and A.8

Parameter	**Value**
Number of antennas	2
Frame errors limit	150

Figures A.7 and A.8 show that the 4.8 format used for 16-QAM is not enough when dealing with 64-QAM. For the chosen SNR interval, 5 integer and 10 fractional bits were sufficient. All simulations are with constant sphere and unsorted QRD.