

A VLSI 8×8 MIMO Near-ML Detector with Preprocessing

Geoff Knagge · Mark Bickerstaff · Brett Ninness · Steven R. Weller · Graeme Woodward

Received: 30 April 2007 / Revised: 1 April 2008 / Accepted: 23 April 2008 / Published online: 1 June 2008
© 2008 Springer Science + Business Media, LLC. Manufactured in The United States

Abstract Multiple-input multiple-output (MIMO) systems are of significant interest due to their ability to increase the capacity of wireless communications systems, but for these to be useful they must also be practical for implementation in VLSI circuits. A particularly difficult part of these systems is the detector, where the optimal maximum-likelihood solution is desirable, but cannot be directly implemented due to its exponential complexity. This paper addresses this challenge and presents a digital circuit design for an 8×8 MIMO detection problem. A key feature is the integrated channel preprocessing unit, which performs the channel decomposition functions that are either omitted or performed “off-line” in other designs. The proposed device achieves near maximum likelihood bit error rate results at 57.6 Mbps. Other novelties include a high speed sorting mechanism and power saving features.

Keywords MIMO · VLSI design · ML detection

1 Problem Background

Multiple-input multiple-output (MIMO) systems utilise spatial diversity between arrays of transmit and receive antennae to achieve high data rates. An existing MIMO device allows for data rates of 28.8Mbps [1] using a quadrature phase-shift keying (QPSK) constellation in a 4×4 configuration. This paper addresses the need to achieve a higher data rate.

There are two methods by which this can be done. One is to increase the constellation size [2], however [3] indicates that in real world cellular systems it would be preferable to first increase the antennae dimensionality.

Both methods increase the size of the decoding problem by an exponential order. In particular, with n transmitters, each transmitting from a constellation of size 2^q , the complexity of the problem becomes $O(2^{qn})$. The MIMO receiver in [1] has a search space size of $2^{2 \times 4} = 256$ and is able to perform a brute force search, but doubling the number of antennae to 8 increases the search space size to 65536, which is beyond currently feasible receiver designs.

To overcome this, lattice decoding, particularly the sphere search variant, is regarded as a very promising candidate for practical, high performance, near-optimal maximum-likelihood (ML) detection algorithms, and applications to MIMO have been studied in [4].

However, the sphere algorithm and the associated preprocessing is still computationally intensive, and optimisations need to be found to further reduce its complexity. In [5], we proposed and analysed a simplified algorithm, similar to the “ k -best” algorithm, with

G. Knagge (✉) · B. Ninness · S. R. Weller
School of Electrical Engineering and Computer Science,
University of Newcastle, Callaghan,
New South Wales 2308, Australia
e-mail: geoff.knagge@newcastle.edu.au

B. Ninness
e-mail: brett.ninness@newcastle.edu.au

S. R. Weller
e-mail: steven.weller@newcastle.edu.au

M. Bickerstaff
National ICT Australia (NICTA), Locked Bag 9013,
Alexandria, New South Wales 1435, Australia
e-mail: mark.bickerstaff@nicta.com.au

G. Woodward
Toshiba Research Europe Limited,
32 Queen Square, Bristol BS1 4ND, UK
e-mail: graeme.woodward@toshiba-trel.com

simulation results indicating that it greatly improved feasibility for VLSI implementation. In [6] we verified that feasibility, and here we expand and improve on those results, presenting a proof of concept device for an 8×8 MIMO application with 30% less area than our previous design.

After introducing the algorithm and summarising its advantages in Section 2, we detail the implementation from Section 3 onward. Section 4 studies the important preprocessing function to find an appropriate numerical system, and preprocessor implementation. Section 5 presents the actual searching mechanism, and Section 6 reveals how the challenging sorting requirements are achieved. Sections 7 to 10 present significant optimisations and discuss results and scalability of the device, while comparing it to existing works.

2 Algorithm

Consider a system model for a MIMO channel with t transmitters and r receivers:

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n}. \tag{1}$$

Here, \mathbf{y} is an r -vector with each element representing the received despread sample from one antenna, \mathbf{s} is the t -vector of transmitted symbols, \mathbf{n} is a length r noise vector, and \mathbf{H} is an $r \times t$ matrix of channel coefficients between antennae.

The MIMO detection problem is to solve Eq. 1 for \mathbf{s} using a channel estimate \mathbf{H} , received symbols \mathbf{y} , and knowledge that the elements of \mathbf{s} are from a finite set of constellation points. The ML detector [7] uses an exhaustive search to find

$$\tilde{\mathbf{s}} = \arg \min_{\mathbf{s} \in \Lambda} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2, \tag{2}$$

$$= \arg \min_{\mathbf{s} \in \Lambda} (\mathbf{s} - \hat{\mathbf{s}})^H \mathbf{H}^H \mathbf{H} (\mathbf{s} - \hat{\mathbf{s}}). \tag{3}$$

Here, Λ is the set of possible decisions over all users, \mathbf{H}^H denotes the conjugate transpose of \mathbf{H} , and $\hat{\mathbf{s}}$ indicates where the received signal vector, \mathbf{y} , lies within the lattice of possible original transmissions. The $\hat{\mathbf{s}}$ vector is the unconstrained ML estimate of \mathbf{s} , given by a multiplication by the pseudoinverse of \mathbf{H} :

$$\hat{\mathbf{s}} = (\mathbf{H}^H \mathbf{H})^{-1} \mathbf{H}^H \mathbf{y}. \tag{4}$$

The lattice decoder reduces the search space complexity via a Cholesky or QR decomposition, resulting in an upper triangular \mathbf{U} such that $\mathbf{U}^H \mathbf{U} = \mathbf{H}^H \mathbf{H}$, with

the added constraint that the diagonals of \mathbf{U} are real and non-negative.

The upper triangular nature of \mathbf{U} allows the optimisation problem to be structured as a tree search. Fig. 1 presents a hypothetical $t = 4$ antennae problem, with binary decisions for each transmitter. Each transmitter is represented by one level of the tree, and the branches on each level represent a choice of one of the constellation points available for that transmitter.

The search begins at the root of the tree, on Level 0, with a zero cost. The decision for the first user is represented by level 1, where the last element of $\hat{\mathbf{s}}$ is assigned the constellation point for a given branch, and the remaining elements are yet to be defined. The branch cost corresponds to the multiplication between the last row of \mathbf{U} and the difference between \mathbf{s} and the current $\hat{\mathbf{s}}$ estimate. Since the undefined elements correspond to zeros in the last row of \mathbf{U} , they are irrelevant.

Similarly, Level 2 represents a decision for the second last element of $\hat{\mathbf{s}}$. Defining cost C_{t+1} as 0, the cost for the node at level $t + 1 - n$ of the tree is

$$C_n = C_{n+1} + \left| u_{nn} (s_n - \hat{s}_n) + \sum_{j=n+1}^t u_{nj} (s_n - \hat{s}_n) \right|^2, \tag{5}$$

with the costs increasing monotonically as the tree is traversed. The leaves at level t of the tree represent the entire collection of decisions, and have a cost that is the sum of the cost contributions associated with each of the branches taken to reach that leaf from the root of the tree.

To decrease search time, multiple branches of the tree are simultaneously traced using k parallel search elements, with all elements being restricted to the same level of the tree to allow ease of implementation. Each element is assigned a single candidate node, of which the cost of the child nodes are evaluated. The best k of all of the candidate child nodes are then selected and used as candidate nodes for the next level of the tree. When the leaf nodes are evaluated, the search is terminated and the best k may be used to generate soft information about the decision for each transmitter.

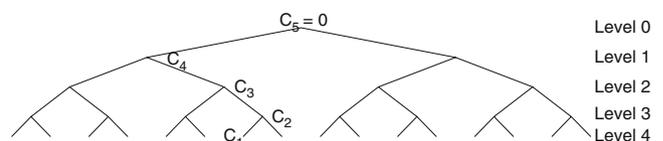


Figure 1 Example of cost allocations for tree search decisions. Note that “Level 0” does not actually exist in the search, since the first decision is represented by “Level 1”.

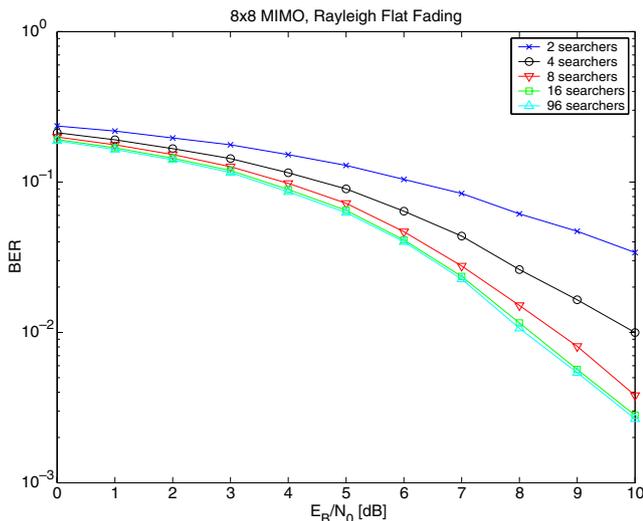


Figure 2 Performance of single-pass approach on a 8 × 8 MIMO system, using QPSK.

This “single pass” tree search method is a simplification of the “Sphere” variant of lattice search algorithms, as described by [4]. The sphere search uses “back-tracking” to search other unexplored tree branches once the leaf nodes are reached and, if a full tree search is allowed, produces the ML result. Usually a “radius” is used to place a limit on the acceptable node cost, with nodes and their branches pruned to reduce the search space if they exceed the radius. Despite this and other optimisations, the sphere search requires an unacceptably large amount of time to properly process the search tree.

However, as we noted in [5], once there are a sufficient amount of parallel search elements, back-tracking rarely finds any lower cost leaf nodes. Therefore, back-tracking has no useful effect and the algorithm can be simplified to the “*k*-best” variant described here.

The implementation benefits of this single-pass approach are significant. Since the search is terminated as soon as the leaf nodes are reached, no back-tracking is required, and so no stack or heap structure is necessary to remember branches that were not followed. In addition, there needs to be no concept of a sphere radius, nor any sorting of the list of best leaves. In

summary, by removing the need for back-tracking, we have eliminated the associated overhead costs that previously limited the feasibility of implementation. In addition, this search strategy allows other implementation optimisations to achieve a more feasible lattice search detector.

To confirm that the single-pass approach does not significantly impact error-rate performance for MIMO systems, a software model of a flat fading MIMO system has been used, with the results shown in Fig. 2. The optimal ML result is not shown, both for clarity and because it requires excessive time to obtain a smooth curve. However, the points that were obtained showed indicated that the 96 searcher curve was almost visually indistinguishable from the ML curve. The results show that, in an 8 × 8 QPSK MIMO system, as few as 8 searchers are needed to obtain a result close to the optimal BER curve, and 16 searchers obtains a very close approximation. Using additional searchers provides little benefit but requires additional circuit complexity. Based on these results, we propose a device using *k* = 16 parallel searchers.

3 Top Level Design

The target timing performance of the design is based on a code division multiple access (CDMA) system using high speed downlink packet access (HSDPA), with 8 antennae and a QPSK constellation. With 15 orthogonal spreading codes in use, this gives a required data rate of 3.6×10^6 symbols per second, or 57.6 Mbps. The chosen clock speed of 122.88 MHz implies the need for three parallel processing engines, to achieve the average throughput of one symbol every 32 clock cycles.

Additionally, to allow for fading channels, the channel estimate may be refreshed as often as every 2048 chips, which requires the channel to be preprocessed on each update. The preprocessing hardware is shared with the search centre generator, so this function needs to be performed as fast as possible.

Figure 3 Top level architecture, showing three parallel search engines interfacing to preprocessor and output logic.

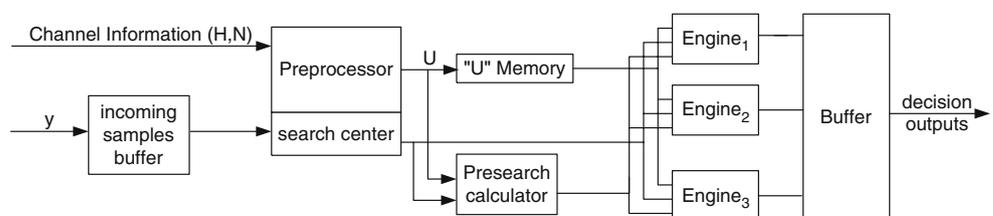


Figure 4 Flow of data through part 1 of the QR decomposition algorithm.

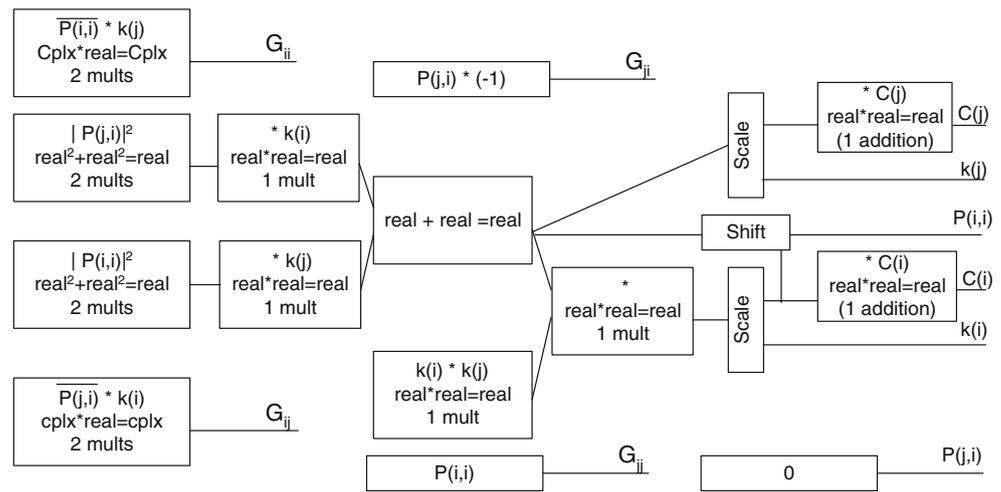


Figure 3 shows an overview of the top level architecture chosen. Channel information is received by the preprocessing unit, where the channel pseudoinverse and decomposition is calculated in preparation for the decoding operations. During this process, incoming samples are held in a circular buffer and released once they are required. Once the preprocessor has finished, it is reused to perform the search centre calculations. The search centre \hat{s} vector is submitted to a presearch unit, with the results queued to the next available search engine in preparation for processing. Once processed, the results are reassembled into their original order, and output sequentially to an external interface.

The following sections consider each of these major components in detail.

4 Preprocessing Implementation

The channel preprocessing functions of the algorithm involve finding the QR decomposition, and the pseudoinverse, of the channel matrix. This is a critical component that is omitted from many 4×4 publications such as [8], and is even more challenging for an

8×8 system. The calculation of the search centre also relies on precomputed values, and in this section we determine the most appropriate methods for determining these values.

4.1 QR Decomposition

The QR decomposition is required to generate the tree structure for the search, and is normally very complex due to its requirement for division and square root operations. This is addressed in [9], where the key concept is a “scaled and decoupled” QR (SDQR) decomposition that separates the numerators and denominators to avoid these difficult operations. For an $m \times n$ channel matrix, \mathbf{H} , its SDQR decomposition is

$$\mathbf{H} = \mathbf{QR} = \Phi^H \mathbf{K}^{-1} \mathbf{P}, \tag{6}$$

where $\mathbf{R} = \mathbf{K}^{-\frac{1}{2}} \mathbf{P}$ and $\mathbf{Q}^H = \Phi^H \mathbf{K}^{-\frac{1}{2}}$. The resultant \mathbf{R} is an $m \times n$ upper triangular matrix, \mathbf{Q} is an $m \times m$ orthonormal matrix ($\mathbf{Q}\mathbf{Q}^H = \mathbf{I}$). The matrices \mathbf{P} and Φ are dimensioned the same as \mathbf{R} and \mathbf{Q} respectively, and \mathbf{K} is a real-valued $n \times m$ diagonal matrix.

Algorithm 4.1 Pseudocode structure of SDQR decomposition

```

For i = 1 to min(n, m) do {
  For j = i+1 to n do {
    Part 1 - Calculate Givens Rotation, as shown in Fig. 4
    Part 2 - Recalculate  $\mathbf{P}(i, j + 1)$  to  $\mathbf{P}(i, n)$  {To find  $\tilde{\mathbf{P}} = \mathbf{P}\mathbf{G}$ }
            - Recalculate  $\mathbf{P}(j, j + 1)$  to  $\mathbf{P}(j, n)$ 
    Part 3 - Recalculate  $\Phi(i, 1)$  to  $\Phi(i, n)$  {To find  $\tilde{\Phi} = \mathbf{G}\Phi$ }
            - Recalculate  $\Phi(j, 1)$  to  $\Phi(j, n)$ 
  }
}
    
```

The SDQR algorithm is detailed in [9], but is summarised in the following discussion of our implementation. Initially, $\mathbf{P} = \mathbf{H}$ and \mathbf{K} and Φ are identity matrices. Our architecture breaks the algorithm into three distinct segments, as outlined in Algorithm 4.1, and as discussed in the following.

4.1.1 Givens Rotation Calculation

This performs the first stage of each iteration, returning the new diagonal element $\mathbf{P}(i, i)$, and the Givens matrix used to correct the affected rows of the \mathbf{P} and Φ matrices. The Givens matrix, \mathbf{G} , is an identity matrix, apart from four entries, $\mathbf{G}_{i,i}$, $\mathbf{G}_{i,j}$, $\mathbf{G}_{j,i}$ and $\mathbf{G}_{j,j}$ that are altered to produce the required rotation. A scaling factor K_i is also produced.

The basic flow of data can be seen in Fig. 4, showing that there are several calculations that are dependent on each other, plus a few that are independent. These independent elements include the Givens rotation matrix elements, \mathbf{G}_{xx} , and so these may be calculated first to allow parts 2 and 3 to proceed in parallel before this part has completed. Using the work timeline provided by Fig. 5, only four work cycles will be needed, regardless of the size of the decomposed matrix, and only four real number multipliers are required.

4.1.2 Rotation of \mathbf{P} and Φ

As seen in Fig. 6, part 2 is simple, requiring four complex multiplications and two additions per column, per iteration. With only four parallel complex multipliers, only one work unit is required for the calculation each column of \mathbf{P} . The rotation of Φ uses exactly the same process as for \mathbf{P} , but with different values.

4.2 Unconstrained ML Estimate

While the channel decomposition only needs to be performed once per channel update, the unconstrained ML estimate, $\hat{\mathbf{s}}$, as defined in Eq. 4, needs to be calculated for every search operation. Six possible approaches that may be taken for this calculation are considered in the following sections.

Cycle 1	Cycle 2	Cycle 3	Cycle 4
$g_{ii} = \overline{P(i,i)} * k(i)$	$X_1 = P(i,i) ^2$	$X_1 * k(i)$ $X_2 * k(i)$	$W=+$
$g_{ij} = \overline{P(j,i)} * k(i)$	$X_2 = P(j,i) ^2$	$P(i,i)=k(j)' = k(i) * k(j)=Z$	$k(i)' = ZW = P(i,i)$

Figure 5 Timeline of calculations for part 1 of the QR decomposition algorithm.

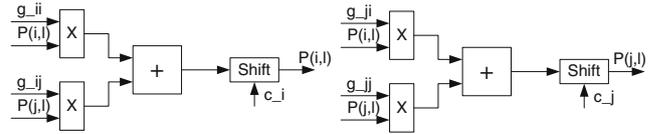


Figure 6 Architecture for recalculation of one column of \mathbf{P} or Φ .

4.2.1 Method 1: Back Substitution Without Φ

Using the SDQR decomposition $\mathbf{H} = \Phi^H \mathbf{K}^{-1} \mathbf{P}$, noting that $\mathbf{Q}\mathbf{Q}^H = \Phi^H \Phi \mathbf{K}^{-1} = \mathbf{I}$, and substituting into Eq. 4,

$$\hat{\mathbf{s}} = (\mathbf{P}^H \mathbf{K}^{-1} \Phi \Phi^H \mathbf{K}^{-1} \mathbf{P})^{-1} \mathbf{H}^H \mathbf{y} \tag{7}$$

$$(\mathbf{P}^H \mathbf{K}^{-1} \Phi \Phi^H \mathbf{K}^{-1} \mathbf{P}) \hat{\mathbf{s}} = \mathbf{H}^H \mathbf{y} \tag{8}$$

$$\mathbf{P}^H \mathbf{K}^{-1} \mathbf{P} \hat{\mathbf{s}} = \mathbf{H}^H \mathbf{y} \tag{9}$$

Since \mathbf{P} is in upper triangular form, then by assigning $\mathbf{x} = \mathbf{K}^{-1} \mathbf{P} \hat{\mathbf{s}}$, \mathbf{x} in Eq. 10 and hence $\hat{\mathbf{s}}$ in Eq. 11 can be solved by back substitution.

$$\mathbf{P}^H \mathbf{x} = \mathbf{H}^H \mathbf{y} \tag{10}$$

$$\mathbf{P} \hat{\mathbf{s}} = \mathbf{K} \mathbf{x} \tag{11}$$

The generic equation for an $M \times M$ system is $\mathbf{A}\mathbf{x} = \mathbf{b}$, with a lower triangular $M \times M$ matrix \mathbf{A} , and with \mathbf{x} and \mathbf{b} as $M \times 1$ vectors. The main complexity issue is that it involves divisions by each of the diagonal elements of \mathbf{A} , which may be precomputed so that only subtractions and multiplications are required for each search centre.

4.2.2 Method 2: Back Substitution Requiring the Calculation of Φ

Alternatively, a single back-substitution may be used to obtain $\hat{\mathbf{s}}$ if the matrix Φ is available.

$$\mathbf{Q}\mathbf{R}\hat{\mathbf{s}} = \mathbf{y} \tag{12}$$

$$\Rightarrow \Phi^H \mathbf{K}^{-1} \mathbf{P} \hat{\mathbf{s}} = \mathbf{y} \tag{13}$$

$$\Rightarrow \mathbf{K}^{-1} \mathbf{P} \hat{\mathbf{s}} = \Phi^{-H} \mathbf{y} \tag{14}$$

$$\Rightarrow \mathbf{K}^{-1} \mathbf{P} \hat{\mathbf{s}} = \mathbf{K}^{-1} \Phi \mathbf{y} \tag{15}$$

$$\Rightarrow \mathbf{P} \hat{\mathbf{s}} = \Phi \mathbf{y} \tag{16}$$

The main difference in complexity between the two back-substitution methods is a trade-off between preprocessing and per-symbol calculations. If the calculation of Φ is allowed during the QR decomposition, then the effort required to calculate $\hat{\mathbf{s}}$ is halved, however additional effort is required to determine Φ .

4.2.3 Method 3: Direct Calculation of $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ with Gauss-Jordan Elimination

The pseudo-inverse may be precalculated for each channel update, and applied to all received vectors sharing the same channel estimate. A practical method of performing a matrix inverse is via Gauss-Jordan elimination, although matrix inverses are generally undesirable for VLSI implementation.

4.2.4 Method 4: Direct Calculation of \mathbf{H}^{-1} with Gauss-Jordan Elimination

When using the Gauss-Jordan inverse method, a more direct approach is to invert the channel matrix instead of computing the pseudo-inverse. The use of this variation is limited to square channel matrices, and is well known to be less numerically stable than other methods. For the calculation of the actual inverse, the algorithm and number of calculations are identical to those in the third method. The computational advantage is that it is not necessary to calculate any additional matrix multiplications.

4.2.5 Method 5: Direct Calculation of $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ by Inverting \mathbf{P}

It is also possible to obtain the pseudo-inverse through inverting the outputs of the QR decomposition and recalling that $\Phi^{-H} = \mathbf{K}^{-1}\Phi$:

$$\mathbf{y} = \mathbf{H}\hat{\mathbf{s}} \quad (17)$$

$$= \Phi^H\mathbf{K}^{-1}\mathbf{P}\hat{\mathbf{s}} \quad (18)$$

$$\mathbf{K}^{-1}\Phi\mathbf{y} = \mathbf{K}^{-1}\mathbf{P}\hat{\mathbf{s}} \quad (19)$$

$$\Phi\mathbf{y} = \mathbf{P}\hat{\mathbf{s}} \quad (20)$$

$$\mathbf{P}^{-1}\Phi\mathbf{y} = \hat{\mathbf{s}} \quad (21)$$

This is similar to the Gauss-Jordan method, except that the triangulation of \mathbf{P} has already been performed by the QR decomposition, and this removes much of the complexity of a standard inverse.

4.2.6 Method 6: Direct Calculation of $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ with Newton-Raphson Method

The Newton-Raphson method of convergence provides a convenient algorithm for obtaining an estimate of an inverse. It is actually a zero finding algorithm for polynomials, and the basic form of the iteration is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

where $f(\cdot)$ is a function for which the zero is required.

The concept can also be extended to matrices. To find the inverse \mathbf{B} of a Hermitian matrix $\mathbf{A} = (\mathbf{H}^H\mathbf{H})^{-1}$, the function to be zeroed is $f(\mathbf{B}_{n+1}) = \mathbf{B}^{-1} - \mathbf{A}$ and the following is performed iteratively:

$$\mathbf{B}_{n+1} = 2\mathbf{B}_n - \mathbf{B}_n\mathbf{A}\mathbf{B}_n \quad (22)$$

$$= \mathbf{B}_n(2\mathbf{I} - \mathbf{A}\mathbf{B}_n) \quad (23)$$

The main advantage of this method is that no divisions are required in the iterations but the disadvantage is the possibility of falling into local minima, or diverging to infinity. However, if the initial values are chosen appropriately, this problem may be minimised. The complexity of this method is determined by the number of iterations that are required.

4.3 Numerical Evaluation

Fixed and floating point simulation models were built for each of the options to determine how accurately they calculate the search centre $\hat{\mathbf{s}}$. For each model, a received signal was calculated in a noiseless simulation, and the mean square error (MSE) from the ideal result measured. The same channels were used for each level of precision and for each calculation method, so any poor channels were common to each technique.

4.3.1 Floating Point Models

The analysis of the floating point model is summarised by Fig. 7, providing several points of comparison. The median is the best indicator of the general trend of the performance of each algorithm, since it is not distorted by a small number of “bad” channels. Such channels may be tolerable if they do not cause frequent instability of the search centre calculation, and this is illustrated by plotting the number of times the MSE exceeds a threshold of 1.

The technique that relies on two back substitutions (not requiring Φ) tends to be a poor performer due to the reliance on previously calculated results. The direct inverse method on $\mathbf{H}^H\mathbf{H}$ is also predictably poor. However, directly inverting \mathbf{H} performs surprisingly well.

The Newton-Raphson based method of search centre calculation also appears to be unsuitable for practical word widths, with the algorithm becoming unstable for an unacceptably high percentage of samples. In fact, for almost all of the techniques, at least 14 bits of precision are required for the MSE to be below the threshold for less than 1% of the time. The exceptions are the method of back-substitution with Φ , and the method of inverting \mathbf{P} .

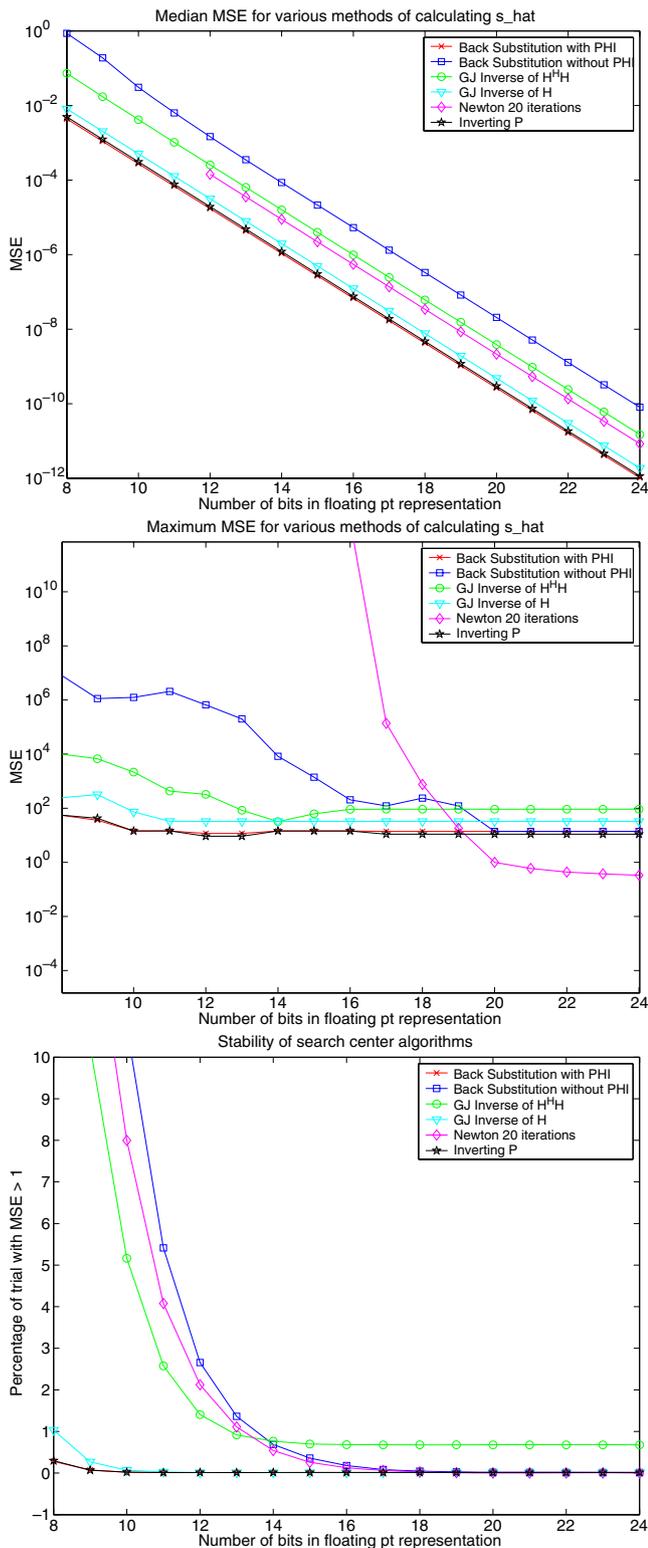


Figure 7 Performance of the search centre calculation techniques under restricted precision floating point constraints.

Based on the floating point systems, the most promising method of calculating the search centre appears to be by inverting the \mathbf{P} matrix result from the SDQR

decomposition. This is convenient, because this is also the technique that requires the least per-symbol calculations, and appears to be quite stable with as little as 12 bits of precision.

4.3.2 Fixed Point Models

Since many of the search centre methods under consideration rely on the result of the QR decomposition, it is necessary to first test the fixed point QR model for feasibility. This test produced the fixed point plot in Fig. 8, which also shows the floating point MSE plot for comparison.

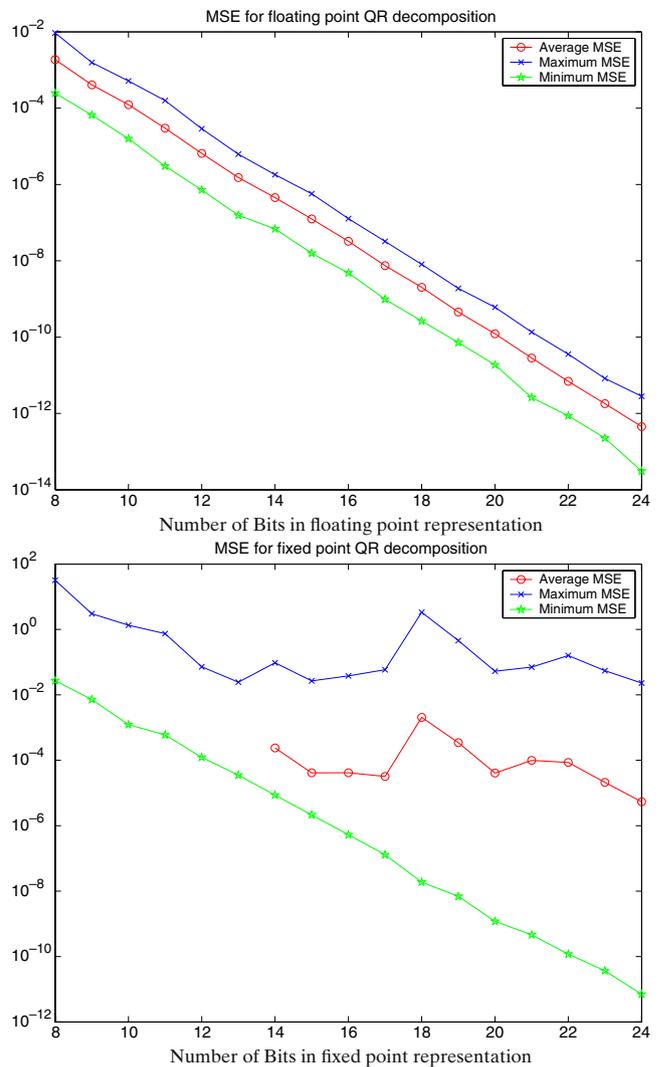


Figure 8 The fixed and floating point MSEs, for the reconstructed channels from 10000 fixed point QR decompositions. The minimum indicates the expected best performance, while uneven fixed point average and maxima lines demonstrate the effect of instabilities and reduced precision.

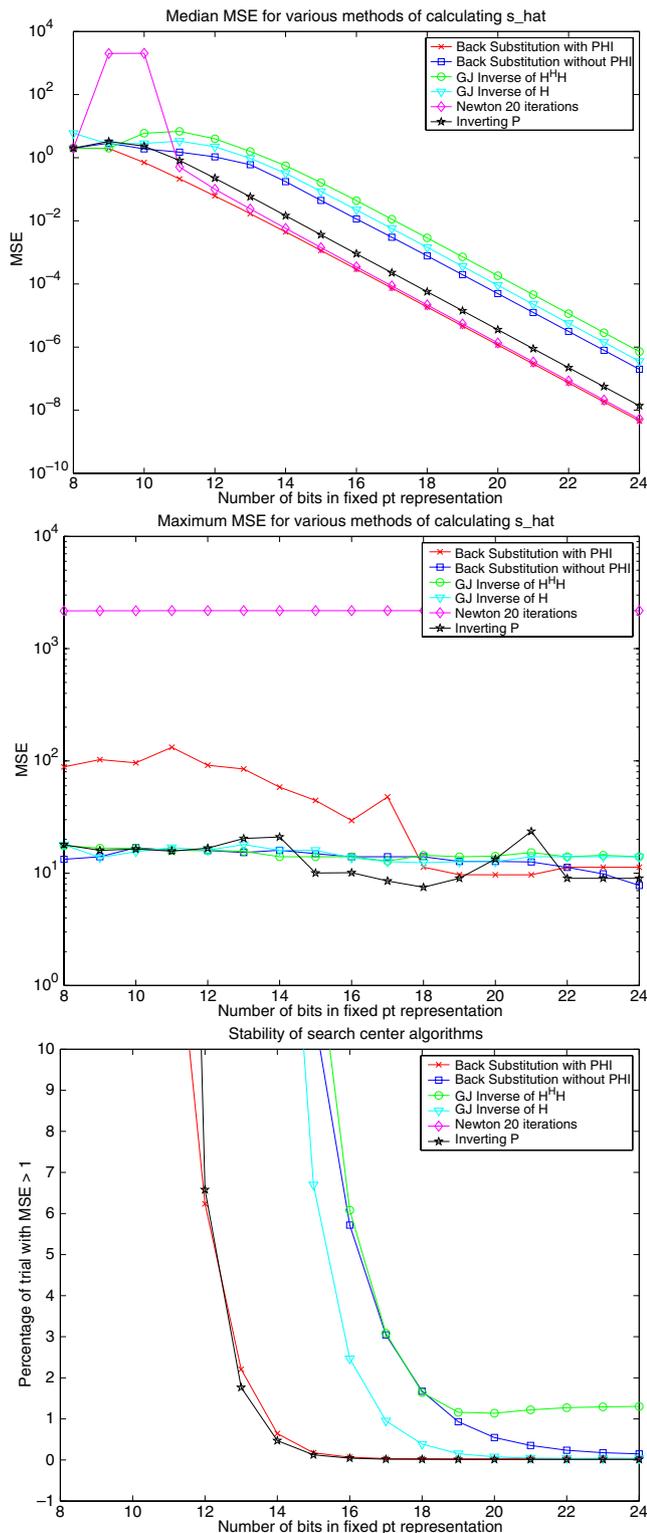


Figure 9 Performance of the search centre calculation techniques under restricted precision fixed point constraints, over 15000 changing channels with 200 iterations each.

While the performance of the fixed point model is reasonable on average, the degradation compared to floating point is significant. These results indicate that

the fixed point model requires at least 16 bits to obtain similar minima and averages to those achieved by the floating point model at only 12 bits.

The search centre tests were repeated on a fixed point system that was restricted by an appropriate set of ranges, resulting in the plots in Fig. 9. The minimum MSEs on these plots confirm that all of the methods are capable of performing well when given enough bits of precision and favourable channels. However, the averages and the maxima are the significant results, and confirm the trends of the floating point models. The method of inverting \mathbf{P} is again the overall best performer.

4.3.3 Fixed vs Floating Point

Fixed point implementations are often considered preferable for VLSI, due to the simplicity of addition operations over floating point. However, it can also carry a heavy numerical cost, since the range is fixed and consequently small values of a particular quantity are represented with relatively little precision. To improve the numerical properties, larger word sizes can be used, but this impacts the physical circuit complexity in terms of size, power, and speed. If, as in this application, a computational unit is to be shared for a variety of calculations, then the ranges of the values used vary greatly from very small to very large. In such systems it is necessary to add flexible range checking and scaling circuitry, which is of similar complexity to that needed for floating point systems.

As seen in Fig. 8, a floating point system offers more stable performance while requiring a smaller word width for similar results. The complicating factor for a floating point system, as opposed to fixed point, is that numbers must be scaled to have the same exponent before any addition operation, and then scale the result so that the mantissa is at least 1 and less than 2. However, when compared with the additional complexity required for a flexible fixed point system, the additional complexity is minimal.

Based on these results, a floating point system with a sign-magnitude format for the mantissa was considered most appropriate for this design. By using a sign magnitude format, the scaling process is simplified and it is easier to detect “zero” values, as only the most significant bit needs to be checked.

The easy detection of zero values allows for lower power requirements. In such cases it is known that the output of some units, such as multipliers, will be zero and so they can be disabled to limit unnecessary logic transitions. The zero value can be achieved by gating the output.

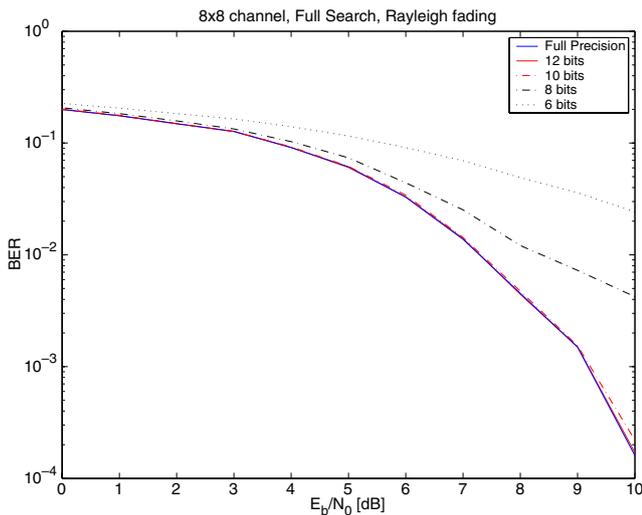


Figure 10 Performance of preprocessor under various bit widths for an 8 × 8 MIMO channel.

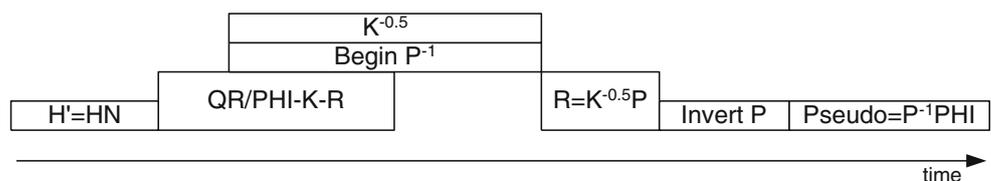
The multipliers normally require 2’s complement inputs, but this can be avoided by always providing positive quantities and, when using Booth recoded multipliers, the sign bits can be used to toggle the appropriate negate signals. The only disadvantage is that, since the multiplier produces 2’s complement outputs, then it is necessary to convert from 2’s complement to sign-magnitude before the rescaling to floating point format.

To determine the necessary word width, a bit-accurate software model of the preprocessor was used. This reveals the effect of varying the mantissa bit size, and Fig. 10 indicates that as little as 10 mantissa bits are necessary to obtain near-optimal performance. However, the final design implements a 12-bit version for greater stability and confidence of a correct result.

4.4 Sequence of Operations

Figure 11 shows a generalised timeline of how the preprocessing functions can be processed. One of the key features of our architecture is a matrix-vector multiplication unit, similar to the one used in [10], which uses an array of multipliers that work in parallel to calculate the product of two vectors in the fastest possible time. However, to do this, they require all elements of the vectors to be available simultaneously. This is achieved

Figure 11 Timeline for execution of preprocessing events.



using wide word-width memories with write-masking that allow individual matrix elements to be written, and entire rows or columns to be read simultaneously. The way that matrices are stored in these memories is a key consideration, and is described in the following.

The following discussion describes the calculations required, and how the matrices may be efficiently stored in this vector format to allow for a fast execution time using the vector multiplier. A summary of this discussion is provided in Table 1, showing how matrices are stored with either rows or columns in each location.

4.4.1 Calculation of Modified Channel Matrix

In some situations it may be desirable to apply a noise whitening matrix \mathbf{N} , as described in [10]. To achieve this, a modified channel matrix is produced by calculating

$$\tilde{\mathbf{H}} = \mathbf{H}\mathbf{N}. \tag{24}$$

The matrix $\tilde{\mathbf{H}}$ is then used as the initial \mathbf{P} matrix for the QR decomposition. To use our matrix-vector multiplication architecture, the \mathbf{H} matrix must be stored with a row vector in each memory location, and the \mathbf{N} matrix requires a column vector in each location.

4.4.2 QR Decomposition

The first stage of QR decomposition requires two values from the same column of the working \mathbf{P} matrix to calculate the Givens Rotation. The second stage also requires column vectors from \mathbf{P} , to compensate the rows affected by the rotation. Therefore, the $\tilde{\mathbf{H}}$ matrix needs to be stored as column vectors. The final stage requires a calculation of

$$\Phi = \mathbf{G}\Phi, \tag{25}$$

where \mathbf{G} is made up of results generated from the first stage. This is effectively the same as the second stage, but instead works on Φ to compensate affected columns. For this to be efficiently calculated using the vector multiplier architecture, the Φ matrix needs to be stored as column vectors.

Table 1 Mapping for use of the data RAMs, with either row or column vectors stored in each location.

Address	RAM1	RAM2
00000 to 00111	H (rows)	N (cols)
01000 to 01111	Φ (cols)	$\tilde{\mathbf{H}}$ (cols)
10000 to 10111	pre- \mathbf{P}^{-1} (cols)	P (cols)
11000 to 11111	U (rows)	\mathbf{P}^{-1} (rows)
		Z (rows)

4.4.3 Scalar Inverse and Reciprocal of Square Roots

The inverse of the diagonals of the final **P** matrix are required to allow for the determination of \mathbf{P}^{-1} , and the reciprocal square root of the elements of **K** are required to calculate **R** from **P**.

4.4.4 Calculation of R Matrix

The **R** matrix may be found using the following calculation :

$$\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P} \tag{26}$$

This result appears on the outputs of the preprocessing unit for storage in external memory.

4.4.5 Calculation of Inverse P Matrix

The inverse of the upper triangular **P** matrix may be found by a partial Gauss-Jordan elimination approach. Since the system is small enough, and some additional time can be spared, the inverse can be calculated by working on one element at a time, and no consideration needs to be given to the method of storing the matrices. An intermediate matrix, Pre-**P** in Table 1, is used in this calculation.

4.4.6 Calculation of Pseudoinverse

The pseudo inverse can be calculated with

$$\mathbf{Z} = \mathbf{P}^{-1}\Phi. \tag{27}$$

This discussion has already constrained Φ to be stored with column vectors in each location, and fortunately that is also convenient for this calculation. This implies that \mathbf{P}^{-1} needs to be stored as row vectors.

4.4.7 Calculation of Search Centre

Once the preprocessing is complete, the search centre may be calculated with

$$\hat{\mathbf{s}} = \mathbf{Z}\mathbf{y}, \tag{28}$$

where the column vector **y** is the received signal, and **Z** is the pseudo-inverse with row vectors stored in each memory address. The resultant $\hat{\mathbf{s}}$ is a column vector that forms the output of the preprocessing engine.

4.5 Preprocessor Architecture

An overview of the primary components and how they interact is illustrated in Fig. 12. The core component is the complex multiplier unit, which is used to implement the vector multiplier architecture, but is also configurable to perform individual multiplications. A dedicated “rotation calculator” performs the Givens rotation stage of the QR decomposition, while the scalar reciprocals and square root operations are performed in separate units. There are two memory blocks in total, to provide storage space for the initial and intermediate data that is required for the preprocessing functions. A state machine control unit coordinates the processing of data between the individual units. The details of the individual components are discussed in the following.

Figure 12 High level representation of the components of the preprocessing unit.

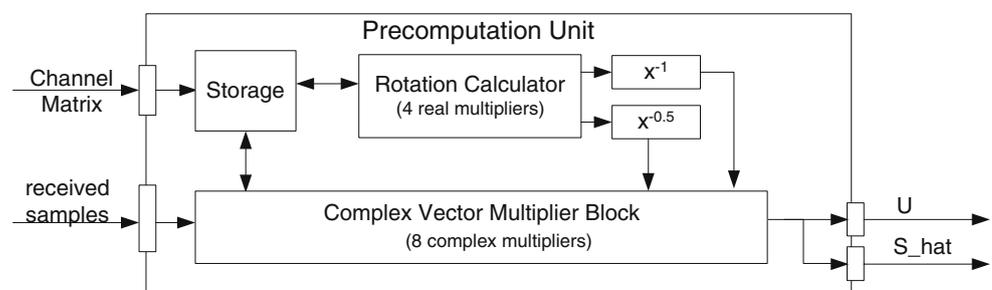
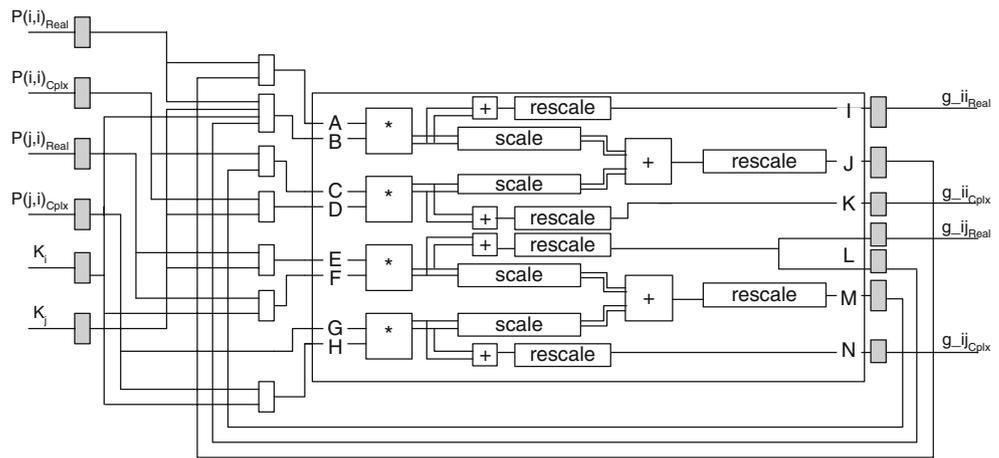


Figure 13 Rotation calculator unit for QR.



4.5.1 Givens Rotation Calculation

The Rotation calculator is presented in Fig. 13, primarily consisting of four real number multipliers and some logic for rescaling floating point values. The sign-magnitude floating point format can be very efficiently implemented, as the K and M outputs in Fig. 13 are known to be positive, and the amount of rescaling required is very limited. For the outputs I, K, L, and N, the signs of the mantissa can be determined by simply examining the signs of the respective inputs. The rescaling required for outputs I, K, L, and N incur little overhead since, with the mantissa constrained to the range $[1 \dots 2)$, the results will be less than 4 and so they merely need to be shifted right by one bit if necessary.

4.5.2 Power Efficient Floating Point Complex Multiplier Unit

The majority of operations involve multiplications between pairs of matrices, or between a vector and a matrix. These are achieved by an array of 8 complex multipliers, using floating point pipelined multipliers to achieve the majority of matrix-vector calculations. These include:

- Individual complex multiplications

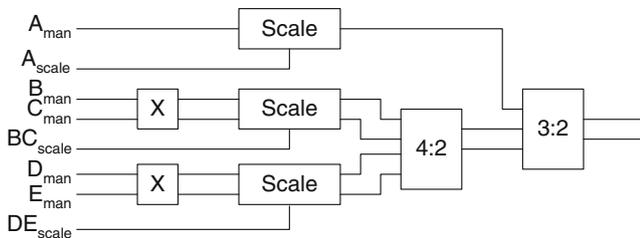


Figure 14 A floating point multiplier pair with an optional offset, A. The output result is $BC + DE + A$.

- Individual complex multiplications, each with an offset added to the result. This is needed for inverting the \mathbf{P} matrix.
- Two pairs of complex multiplications, with their results summed, to perform the calculations for adjusting \mathbf{P} and Φ after the rotations.
- The results of all complex multiplications summed to produce a single output, for vector multiplications

The details of a multiplication unit with an offset input is shown in Fig. 14. It contains a pair of Booth recoded real multipliers, with all arithmetic additions performed in carry-save format with 4:2 and 3:2 adders.

4.5.3 Scalar Division and Inverse Square Root

Divisions are required for inverting \mathbf{P} , and the reciprocal of a square root is used to calculate $\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P}$. This can be achieved with the bisection algorithm, which assumes the existence of a solution between two points, a and b . On each iteration a midpoint, $c = \frac{a+b}{2}$, is selected. The calculation then determines whether the

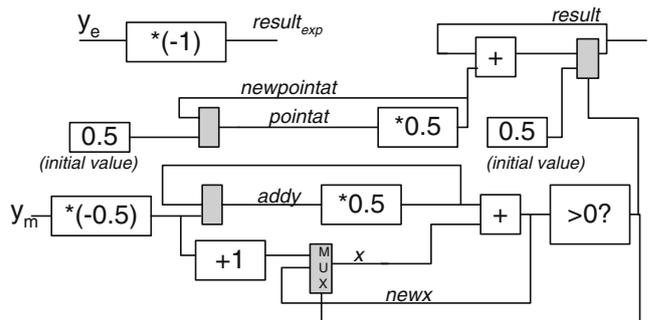
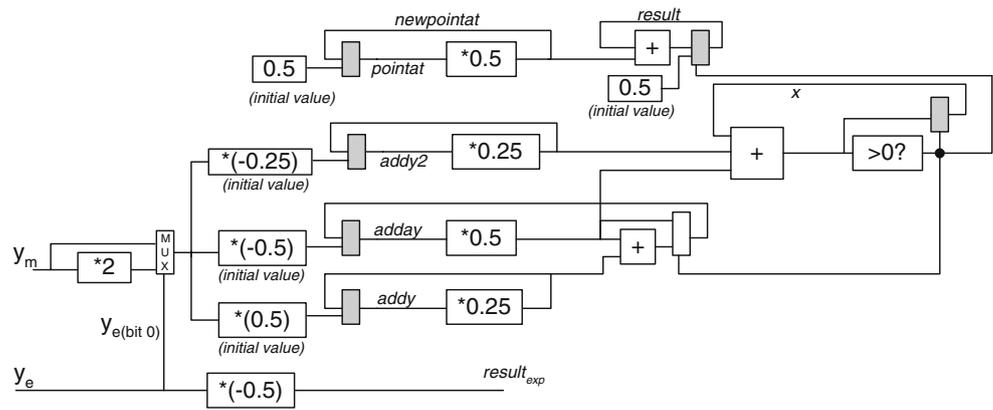


Figure 15 Design of the scalar inverse unit. The greyed boxes represent registers used in the iterative calculation of the result.

Figure 16 Inverse square root unit. Shaded boxes represent registers used to hold results between iterations.



solution is between a and c , or b and c , and either a or b is replaced with c as appropriate.

For the case of finding the inverse of a scalar x , for any candidate point y this involves calculating:

$$\text{cost} = 1 - xy$$

For an appropriate a and b , cost_a would be positive and cost_b would be negative. The midpoint, cost_c , would replace a if it were positive, or replace b if it were negative. As the solution converges to the correct answer, both costs would converge to 0.

Similarly, finding the inverse square root of x , for any candidate point y , involves calculating:

$$\text{cost} = 1 - xy^2$$

In a floating point system, it is known that x_m is between $1.00 \dots 00_2$ and $1.11 \dots 11_2$. Therefore, y_m will be between $0.10 \dots 00_2$ and $1.00 \dots 00_2$ for both the inverse and inverse square root operations.

For each iteration, decisions are made about an individual bit of the result. The test case C is the current result, with a 1 added to the left as a new least significant bit. If the cost of C is positive, then this decision is kept, otherwise that bit is set to zero and the previous decision is retained.

The calculation of costs may also be greatly simplified. Assuming the result so far is A , with a cost of x . i.e.,

$$x = 1 - A^2y,$$

then the cost of adding 0.01_2 is

$$x_{\text{next}} = 1 - (A + 0.01_2)^2y \tag{29}$$

$$= x - 0.10_2Ay - 0.001_2y \tag{30}$$

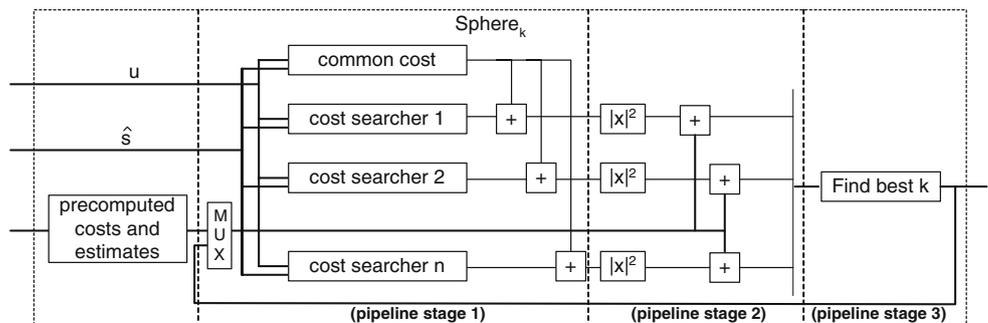
This is significant because the multiplications by 0.10_2 and 0.001_2 are virtually free, and the multiplication Ay can be cheaply obtained in a similar manner. Therefore, the cost of each iteration is little more than a series of additions, and the result may be found quite easily.

A similar, but even more simplified, approach may be taken for finding the inverse of a number. In that case, the cost of adding 0.01_2 is

$$x_{\text{next}} = x - 0.01_2y \tag{31}$$

The use of the bisection style approach is restricted to cases where the function curve between the chosen start points is continuous with no changes in sign of the gradient. These criteria are met in these cases and this

Figure 17 Architecture of a single search engine.



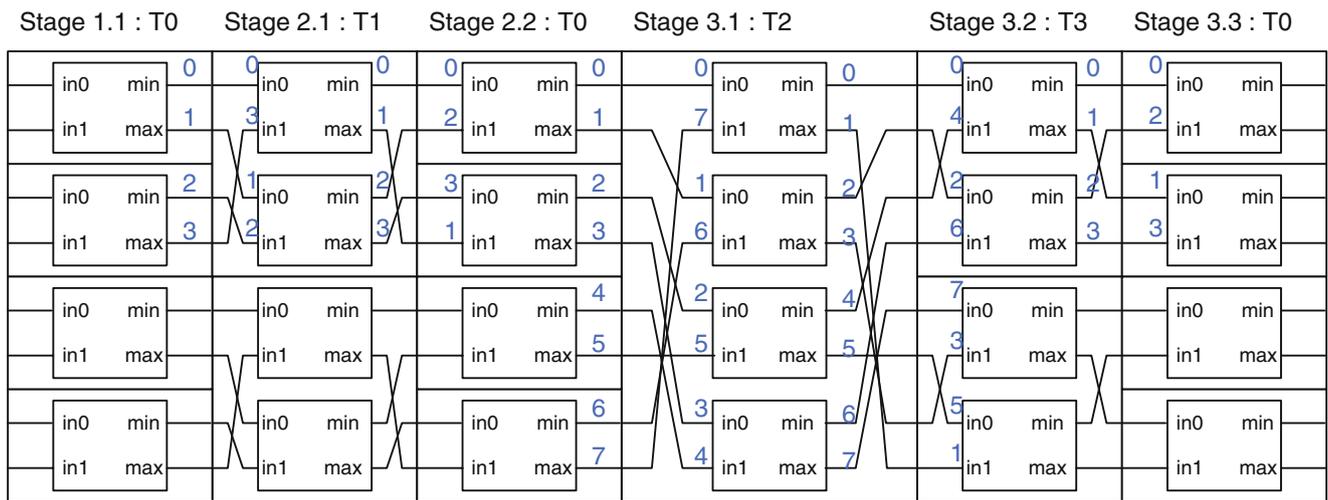


Figure 18 8 input sorting network configuration. The *numbers* indicate the matching inputs and outputs of connecting stages for clarity.

algorithm allows for a very small implementation, as shown in Figs. 15 and 16.

5 Search and Presearch Engines

The function of the search engines is to perform the lattice search operation using the algorithm described in Section 2, by calculation of Eq. 5. This is calculated for each of the children of the k currently selected nodes in the search tree, of which the best k of those children are selected for the next iteration. As described in the following, these are calculated in a series of parallel search engines, and one specially tuned engine that is aimed at calculating the initial stages of the search.

5.1 Search Engine Structure

The search engines each contain a three stage pipeline, which iterates to execute the final six levels of the parallel tree search. The content of these engines is illustrated in Fig. 17, and implements a series of simple node cost calculations in floating point format to evaluate Eq. 5 and expand each successive level of the tree. Once these are calculated, a sorter block is used to determine the best k of these generated options.

The first stage involves tallying up to 16 cost components from the summation of Eq. 5, and the second calculates the $|x|^2$ part of this cost. Finding $|x|^2$ for a complex number x requires only two real multiplications, and so 4 multipliers are required per engine to sequentially achieve this for 16 searchers within 8 double-cycles.

The task of finding the four child costs is also less costly than it may initially seem, since the estimates of

s differ by only the element corresponding the current level of the search. Therefore, only one tally needs to be kept for all child nodes.

Figure 17 omits a hidden stage that is specific to the decoder and not the individual searchers, and that involves calculation the contribution of \hat{s} to the cost. Since the \hat{s} component of the cost is independent of the working estimate, it is common to all searchers within the same engine, and so only needs to be calculated once. A compact implementation of this calculation may be obtained with a single complex multiplier that sequentially tallies the relevant components of the cost on each iteration of the search.

5.2 Presearch Engine

The search engines are optimised for the worst possible case, which involves expanding the 64 children of up to 16 current candidate notes, and then performing a sort to obtain the best 16. However, in the initial stages of the search, the engines do not operate at their full capacity. The first stage simply involves expanding the four branches from the root of the tree, and no sorting is required. The second stage expands these branches one further level, producing 16 candidates. Furthermore, as only four branches are being expanded, much less of the engine’s computation units are required for these stages.

To exploit this, a presearch unit is implemented and optimised to calculate the first two stages of the search in an efficient manner. Without using the presearch unit, the engines would require 8 iterations to perform a search, instead of 6. This would increase their operating time, and hence throughput, by 33%. Consequently, additional engines would be required, increasing the

Table 2 Table indicating which outputs from the previous sorting substage are used as inputs for the next stage.

Input	Stage 2		Stage 3			Stage 4				Stage 5					Stage 6					
	1	2	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	3	2	7	4	2	15	8	4	2	31	16	8	4	3	63	32	16	8	4	2
2	1	1	1	2	1	1	2	2	1	1	2	2	2	1	1	2	2	2	2	1
3	2	3	6	6	3	14	10	6	3	30	18	10	6	2	62	34	18	10	6	3
4			2	7		2	4	1		2	4	4	7		2	4	4	4	7	
5			5	3		13	12	5		29	20	12	3		61	36	20	12	3	
6			3	5		3	6	3		3	6	6	5		3	6	6	6	5	
7			4	1		12	14	7		28	22	14	1		60	38	22	14	1	
8						4	15			4	8	1			4	8	8	1		
9						11	7			27	24	9			59	40	24	9		
10						5	13			5	10	3			5	10	10	3		
11						10	5			26	26	11			58	42	26	11		
12						6	11			6	12	5			6	12	12	5		
13						9	3			25	28	13			57	44	28	13		
14						7	9			7	14	7			7	14	14	7		
15						8	1			24	30	15			56	46	30	15		
16										8	31				8	16	1			
17										23	15				55	48	17			
18										9	29				9	18	3			
19										22	13				54	50	19			
20										10	27				10	20	5			
21										21	11				53	52	21			
22										11	25				11	22	7			
23										20	9				52	54	23			
24										12	23				12	24	9			
25										19	7				51	56	25			
26										13	21				13	26	11			
27										18	5				50	58	27			
28										14	19				14	28	13			
29										17	3				49	60	29			
30										15	17				15	30	15			
31										16	1				48	62	31			

Inputs are then grouped pairwise for compare-and-swap operations. Note that the first stage is not represented here, and only the top half is shown.

circuit area significantly. Therefore, this simple optimisation provides one of the most significant benefits to the overall design.

6 Sorting

The most challenging aspect of the design is to build a sorter that selects the best 16 of the 64 evaluated nodes in one pipeline stage, containing 16 cycles at 122.88 MHz. One of the key features exploited in our solution is that the outputs do not need to be fully sorted, as we only require the best 16 entries in any order. For this purpose, a Batcher sort [11], optimised for this application, was chosen.

Figure 18 illustrates how the sorting network is divided into stages, in which a subset of numbers is fully sorted. The first stage sorts pairs of inputs, the second stage sorts 2² inputs, the third sorts 2³, and so on. Within each stage, a series of pairwise comparisons are made to gradually reorder the elements in a set of substages.

Figure 18 also identifies the inputs by numbers, corresponding to an output of the previous substage. By tabulating these numbers, a series of patterns can be observed, providing a generalisation that allows a sorting system of any size to be designed:

- For the first substage, on sorter element *k* and stage *n*, the two inputs are numbered *k* and 2^{*n*} − *k* − 1.

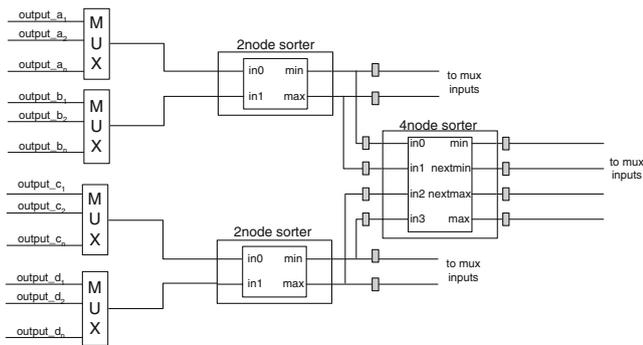


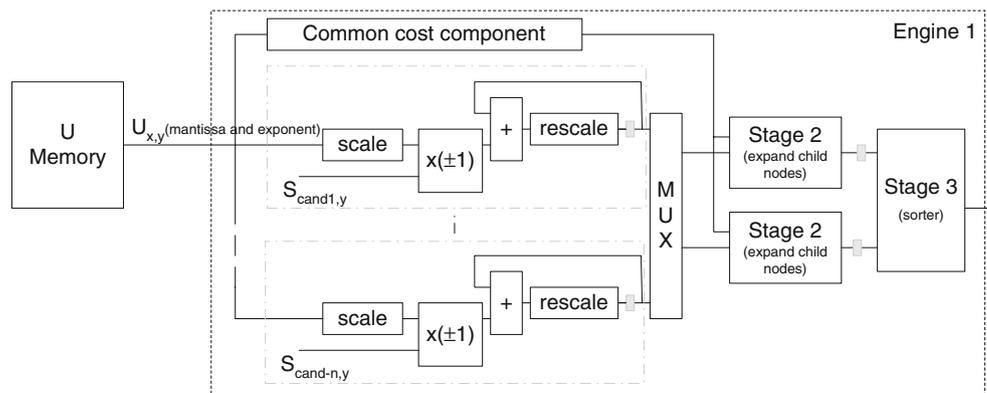
Figure 19 Optimised structure of the sorting elements. The 2-node elements implement the majority of the stages, using multiplexers to switch the inputs, and the 4-node unit implements the combined T3/T0 stage. The *grey boxes* represent registers.

- For the second substage, the sorter elements are divided into two halves. For elements $k = 0$ to $2^{n-2} - 1$, the inputs are numbered $2k$ and $2k + 2^{n-2}$. For elements $k = 2^{n-2}$ to $2^{n-1} - 1$, the inputs are numbered $2^n - 1 - 2k$ and $2^{n-1} - 1 - 2k$.
- Each substage thereafter divides the outputs into two halves, and then applies the second substage rule to each half

When tabulated, as in Table 2, it can be seen that certain patterns of connections repeat. In particular, for each stage $K > 3$, the first three substages are unique and the remaining substages are the same as the corresponding final substages for the previous stage. To sort the 64 inputs, only 13 types of configurations are required to sort the numbers in 22 substages.

To reduce the number of substages to 14, we first note that the final 3 substages are not needed because the set does not need to be fully sorted. The T3/T0 substages are recognised as a repeating combination and combined into one unit, and the first three substages are precomputed as the cost data is generated by the search engine. The resulting sorter architecture is an array of 16 of the 2-node sorter pairs shown in Fig. 19.

Figure 20 Overview of the original search engine architecture, detailing the components that were modified. The *grey boxes* represent registers, clocked every second cycle.



To reduce power, duplicate registers are placed on the outputs of the 2-node elements. If the next stage is a T3/T0 stage, then only the registers connecting to the 4-node element need to be clocked. This blocks logic transitions to the multiplexers and 2-node elements, which will not be used in the next stage. Otherwise, when the next stage is not the combined T3/T0 stage, power is saved by blocking logic transitions from entering the unused 4node element.

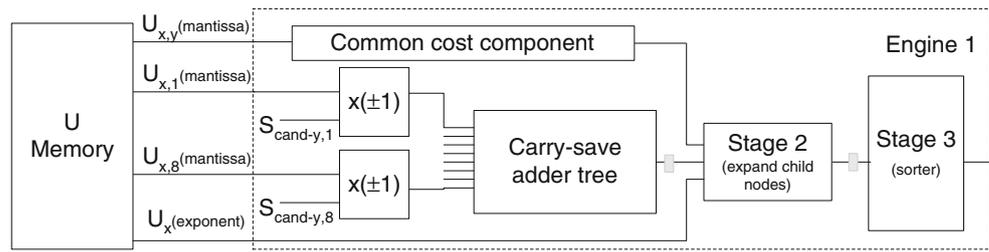
7 Further Optimisation

In our initial publication of this design [6], we claimed a circuit that required a core area of 12mm^2 , and a gate count of 500K cells. However, this has since been reduced to an estimate of 8mm^2 and 350K cells. This 30% reduction in area has been achieved through optimisation of the search engines.

To understand how this optimisation is possible, we consider the architecture of the original design, in Fig. 20. This used floating point representations throughout the entire architecture and used 8 double-length cycles to process each stage of the engine. During Stage 1, each of these double-cycles added one element of the cost tally for each of the 16 candidates. These additions require the two values to be scaled to a common exponent, added, and then rescaled back to floating point form. At the conclusion of Stage 1, these tallies for all 16 candidates were passed on to Stage 2, requiring 16 sets of registers. On each double-cycle of Stage 2, a pair of these candidates were selected and the final costs added to the tally to expand the four child nodes. These were passed onto the sorter in preparation for Stage 3.

Two key changes were made to achieve the reduced area, resulting in the architecture in Fig. 21. The first is through the use of a pseudo-floating point representation for the values of the \mathbf{U} matrix. Instead of each

Figure 21 Overview of the optimised engine architecture, providing a generalised indication of the changes that were made. The registers marked by *grey boxes* are clocked every cycle.



element of the matrix consisting of its own mantissa and exponent, the representation was changed so that the maximum exponent of all of the values in that row was applied to all other elements in that row. This means that all elements in any given row of the matrix were prescaled to use the same exponent. The advantage offered by this is that it is simpler to add cost components to the progressive tally, as no floating point scaling needs to be done before or after the addition. One of the advantages that makes this algorithm highly attractive is the ability to reduce multiplications, and this optimisation allows that benefit to be fully exploited.

The second optimisation, which is achievable after implementation of the first, is by reducing the number of registers between stages and reorganising the order in which cost tallies are calculated. Instead of calculating these costs simultaneously for all candidates over 8 double-cycles, the new design calculates the entire cost tally for each candidate in one single cycle. The new design only calculates one candidate per cycle, allowing for a reduction in duplicated circuitry. After calculation, these are passed directly to Stage 2, which finalises the cost calculation as before, and passes the result to Stage 3 in preparation for the start of the next search operation.

The main advantages gained by this is that there is only one set of cost information being passed from Stage 1 to Stage 2, instead of one for each of the 16 candidates. This is possible because, using the pseudo-floating format, it is very simple to create an adder tree for the 16 possible cost components, considering both

imaginary and real parts. Additionally, since Stage 1 is only working on one candidate at a time, there only needs to be one set of adder circuitry instead of 16. Furthermore, Stage 2 operates on only one candidate at a time, instead of 2, so its complexity is halved.

The results are also based on the use of $k = 16$ parallel search candidates, however Fig. 2 indicates that $k = 8$ may provide an equally acceptable result. The effect of this would be to almost halve the circuitry required for some of the core engine components, reducing the circuit area by as much as an additional 2mm².

Furthermore, the device presented here is a worst case result that uses 12-bits, even though our simulation results indicate that as little as 10 bits may be sufficient. The suitability of a 10 bit version would depend on the individual application, but this has the potential to offer further substantial area savings.

8 Scalability

The preprocessing architecture is easily scalable and is quite suitable for even larger matrices, but will require a longer execution time. Most of the additional effort occurs from the need to perform more iterations to complete the decomposition, with each iteration increasing only linearly in complexity.

The remainder of the architecture is very modular, and so can be resized as required. If higher throughput is required, more engines may be added. If a higher constellation is required, additional search elements

Table 3 Comparison of MIMO implementation proposals.

Garrett et al. [1] performs a full ML search without using a tree search, and so does not require a preprocessor.

Reference	Proposed	[8]	[2]	[1]
Preprocessor	20 K Gates	No	No	N/A
Antennae	8 × 8	4 × 4	4 × 4	4 × 4
List decoder	Yes	No	Yes	Yes
Constellation	QPSK	16QAM	16QAM	QPSK
Throughput, 1 dB	> 57.6 Mbps	≪ 50 Mbps	106 Mbps	28 Mbps
Throughput, 5 dB	> 57.6 Mbps	< 50 Mbps	106 Mbps	28 Mbps
Gates	350 K	117 K	97 K	170 K
Clock Speed	122 MHz	51 MHz	200 MHz	122 MHz

may be necessary, but the basic structure would remain unchanged.

9 Results and Comparison

A prototype device has been prepared and routed (Fig. 22) for a $0.13\mu\text{m}$ technology, with a core size of approximately 8mm^2 for our worst case configuration. By using $k = 8$ searchers and 10 bit word widths, we predict this could be reduced to below 6mm^2 . This establishes the feasibility of 8×8 MIMO detection, with a unique integrated preprocessing unit that is commonly omitted from simpler 4×4 designs. The algorithmic performance of our proposal indicates near-ML results.

Table 3 compares the proposed design with existing 4×4 MIMO implementations. However, it is difficult to achieve a fair comparison since the 8×8 system contains more levels in the tree, resulting in a more complex problem with more stages of decoding operation. The advantage is that, while a 4×4 may have identical search space and raw throughput as the proposed design, the choice of higher antennae dimensionality over higher constellation size is likely to provide better performance in practical applications [3].

Sphere based devices, such as [8], claim to have a lower cell area, but the throughput at low signal-to-noise ratios (SNRs) is significantly impaired. Additionally, [8] does not provide soft decisions required for high performance error detection algorithms. Low

SNRs are typical of realistic cellular channels, so the k -best style of algorithm has a clear advantage of having a constant throughput irrespective of SNR.

Hochwald and ten Brink [4] estimates that the complexity increases at best in $O(x^3)$ with tree depth x , so if sphere decoder designs such as [8] were scaled to match the same complexity and throughput, the size would be more comparable. However, as illustrated in [5], the key advantage of this algorithm is the reduced number of calculations performed.

The soft output decoder in [2] claims the capability of a very high throughput, but this is beyond the capabilities of the HSDPA standard targeted here. While their design is simplified through ordering based on SNR and channel information, this does not appear to scale well to 8×8 systems. Without this ordering, our design and [2] both require up to three times as many soft candidates, accounting for much of the difference in area.

In [12] it is claimed that standard sphere detection provides similar bit error rate and throughput with a lower silicon area. However, it appears that this is referring to a 4×4 system and it is not clear what particular operating point is targeted. Our own experiments suggest that such claims are dependant on particular configurations and, when extended to an 8×8 system matching the performance of our proposed device, these claims would not hold.

10 Conclusion

We have presented a unique scalable device based on a k -best algorithm, confirming the feasibility of decoding data from an 8×8 MIMO system at 56 Mbps for all SNRs, and significantly improving our previous results. It includes the crucial channel preprocessing functionality, 16 candidates for soft output data, and addresses the high speed sorting challenge that is characteristic of this algorithm. We have addressed the many difficulties in implementing this algorithm, and offered possibilities for even further optimisation.

References

1. Garrett, D., Davis, L., ten Brink, S., Hochwald, B., & G. Knagge (2004). A 28.8 Mb/s 4×4 MIMO 3G high-speed downlink packet access receiver with normalized least mean square equalization. In *IEEE international solid-state circuits conference (ISSCC)* (pp. 420–421) (February).
2. Guo, Z., & Nilsson, P. (2006). Algorithm and implementation of the k -best sphere decoding for MIMO detection. *IEEE*

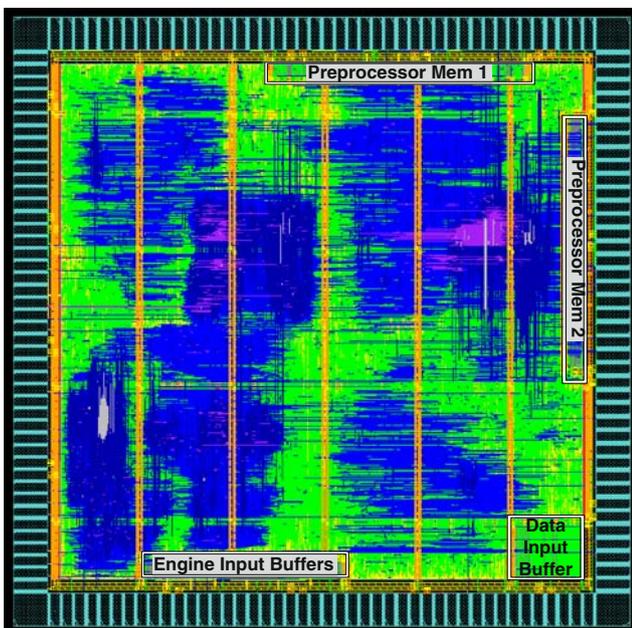


Figure 22 Plot of the chip design. The boxed areas indicate the memory blocks, illustrating the relative size to the core logic area.

- Journal on Selected Areas in Communications*, 24, 491–503 (March).
3. Davis, L. M., Garrett, D. C., Woodward, G. K., Bickerstaff, M. A., & Mullany, F. J. (2003). System architecture and ASICs for a MIMO 3GPP-HSDPA receiver. In *57th IEEE semiannual vehicular technology conference, 2003. VTC 2003-Spring* (Vol. 2, pp. 818–822) (April).
 4. Hochwald, B., & ten Brink, S. (2003). Achieving near-capacity on a multiple-antenna channel. *IEEE Transactions on Information Theory*, 51, 389–399 (March).
 5. Knagge, G., Woodward, G., Ninness, B., & Weller, S. (2004). An optimised parallel tree search for multiuser detection with VLSI implementation strategy. In *IEEE global telecommunications conference, 2004. (GLOBECOM '04)* (pp. 2440–2444) (December).
 6. Knagge, G., Bickerstaff, M., Ninness, B., Weller, S., & Woodward, G. (2006). A VLSI 8×8 MIMO near-ML decoder engine. In *IEEE workshop on signal processing systems design and implementation (SiPS06)* (pp. 387–392) (October).
 7. Verdú, S. (1986). Minimum probability of error for asynchronous Gaussian multiple-access channels. *IEEE Transactions on Information Theory*, 32, 85–96 (January).
 8. Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., & Bolcskei, H. (2005). VLSI implementation of MIMO detection using the sphere decoding algorithm. *IEEE Journal of Solid State Circuits*, 40, 1566–1577 (July).
 9. Davis, L. M. (2003). Scaled and decoupled cholesky and QR decompositions with application to spherical MIMO detection. *IEEE wireless communications & networking conference (WCNC)* (pp. 326–331) (March).
 10. Knagge, G., Garrett, D. C., Venkatesan, S., & Nicol, C. (2003). Matrix datapath architecture for an iterative 4×4 MIMO noise whitening algorithm. In *ACM great lakes symposium on VLSI circuits (GLSVLSI)* (pp. 590–593) (April).
 11. Sharma, N. K. (1997). Modular design of a large sorting network. In *Third international symposium on parallel architectures, algorithms, and networks, 1997. (I-SPAN '97)*. (pp. 362–368) (December).
 12. Burg, A., Borgmann, M., Wenk, M., Studer, C., & Bolcskei, H. (2006). Advanced receiver algorithms for mimo wireless communications. In *Proc. design automation and test in Europe conf. (DATE)* (March).



Geoff Knagge received the B Comp. Sci. degree (2001), BE (Hons I) in computer engineering (2002), and Ph.D. (2007) from the University of Newcastle, Australia. His Ph.D. thesis investigated algorithms for wireless digital communications with a focus on feasibility of implementation in VLSI circuits. He currently works for the Signal Processing and Microelectronics group at

the University of Newcastle, which focuses on algorithms and electronic design for communications and control applications. He is an author on 2 journal articles, 8 published conference papers, 1 US patent, and an additional pending patent.



Mark A. Bickerstaff (S'87 - M'94), BSc (Hons) computer science ('85), BE in electrical engineering ('87) University of Sydney, Australia; Ph.D. (1994) University of New South Wales, Australia; From 1993 to 1999, he worked as a Research Engineer on DSP IP-cores for OFDM and IC Design flows in the Department of Electronics at Macquarie University, Australia. He was involved in the creation of 5 integrated circuits. From 1999 to the 2003, he worked at Bell Labs Research, Lucent Technologies, in the Sydney-based wireless research team on IP cores for cellular mobile and was involved in the creation of 5 chips. From 2003 to 2007, he worked at the Australian Design Centre of Agere Systems (now LSI Corporation) designing digital baseband architectures for cellular mobile systems and was involved in the design of 2 chips. From 2007 he has worked at National ICT Australia (NICTA) in gigabit wireless systems. He has 21 journal/conference publications and 7 US patents with 5 patents pending.



Brett Ninness was born in 1963 in Singleton, Australia and received his BE, ME and Ph.D degrees in Electrical Engineering from the University of Newcastle, Australia in 1986, 1991 and 1994 respectively. He has stayed with the School of Electrical Engineering and Computer Science at the University of Newcastle since 1993, where he is currently a Professor.

His research interests are in the areas of system identification and stochastic signal processing, in which he has authored approximately one hundred papers in journals and conference proceedings. He has served on the editorial boards of *Automatica*, *IEEE Transactions on Automatic Control* and is currently Editor in Chief for *IET Control Theory and Applications*.



Steven R. Weller was born in Sydney, Australia, in 1965. He received the B.E. (Hons I) degree in Computer Engineering in 1988, the M.E. degree in Electrical Engineering in 1992, and the Ph.D. degree in Electrical Engineering in 1994, all from the University of Newcastle, Australia. From April 1994 to July 1997 he was a Lecturer in the Department of Electrical and Electronic Engineering, at the University of Melbourne, Australia. In July 1997 he joined the University of Newcastle, where he is currently an Associate Professor and Head of School for the School of Electrical Engineering and Computer Science. His current research interests include low-density parity-check codes, iterative decoding algorithms, and space-time coded communications.



Graeme K. Woodward (S'94-M'99) was awarded B.Sc and BE and Ph.D. degrees from The University of Sydney in 1990, 1992 and 1999 respectively. His experience includes digital switching systems with Alcatel Australia 1991–4 and channel modelling, 3G cellular systems and multiuser detection for CDMA, with SP Communications in 1999–2000 in collaboration with The University of Oulu, Finland.

In 2001 he joined Bell Laboratories Research, Lucent Technologies, researching 3G systems, multi-antenna systems, multi-user detection, interference suppression and adaptive digital filter algorithms for equalisation. With the Bell Labs group transitioning to Agere Systems Australia and subsequently LSI Logic in 2003 and 2007 respectively, the focus moved to terminal-side algorithms for High Speed Downlink Packet Access (HSDPA) and Long Term Evolution (LTE) targeted at VLSI implementation. In 2008 Dr Woodward moved to the Telecommunications Research Laboratory, Toshiba Research Europe.

Dr Woodward has authored seven journal articles, more than 20 conference papers and is an inventor on numerous US patent applications. He has served on several conference committees, including VTC'06, ISSSTA04 and multiple AusCTW workshops.