A VLSI 8x8 MIMO Near-ML Decoder Engine

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

***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 decoder, where the optimal maximum-likelihood (ML) solution is desirable, but cannot be directly implemented due to its exponential complexity.**

The paper presents the first published 8×8 MIMO detection engine with an integrated channel preprocessing unit, achieving near-ML BER results at 57.6Mbps, using QPSK in an extended HSDPA application. Other novelties include the high speed sorting mechanism and power saving features.

I. 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 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 a real world cellular system 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 $O\left(2^{qn}\right)$. 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 to 65536, which is beyond currently feasible receiver designs.

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

However, the 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 simulation results indicating that it greatly improved feasibility for VLSI implementation. Here, we progress to a proof of concept device for an 8×8 MIMO application.

G. Knagge, B. Ninness, and S. Weller are with School of Electrical Engineering and Computer Science University of Newcastle, Callaghan NSW 2308, Australia {gknagge,brett,steve}@ee.newcastle.edu.au

M. Bickerstaff and G. Woodward are with Agere Systems, Lvl 3, 11-17 Khartoum Rd, North Ryde NSW 2113, Australia {markb,graemew@agere.com}

II. 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 (1) for $\mathbf{s}$ using a channel estimate $\tilde{\mathbf{H}}$, the received symbol estimates $\mathbf{y}$, and knowledge that the elements of $\mathbf{s}$ are from a finite set of constellation points. The maximum-likelihood (ML) detector [6] uses an exhaustive search to find

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

where Λ is the set of possible decisions over all users. This paper uses the lattice decoder approach of expressing (2) as

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

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

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

A lattice decoder reduces the search space complexity via a Cholesky or QR decomposition, resulting in an upper triangular $\mathbf{U}$ such that $\mathbf{U}^H\mathbf{U} = \tilde{\mathbf{H}}^H\tilde{\mathbf{H}}$, 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, with each transmitter representing one level of the tree, and the branches representing a choice of one of the constellation points available for each transmitter. Each leaf then represents the entire collection of decisions, and has 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.

Fig. 1 presents an example for a $t = 4$ antennae problem, with binary decisions for each transmitter. The first decision is represented by level 1, which corresponds to the last row of $\mathbf{U}$. Defining cost C_{t+1} as 0, the cost for the node at level $t + 1 - n$ of the tree is

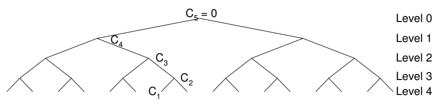


Fig. 1. Example of cost allocations for tree search decisions. Note that "Level 0" does not actually exist in the search, since the first decision is represented by "Level 1".

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

with the costs increasing monotonically as the tree is traversed.

On each level of the tree all of the children of the current candidate nodes are evaluated, and the best k of these are kept as candidate nodes for the next iteration. When the leaf nodes are evaluated, the best k are selected and these can be used to generate soft information about the decision for each transmitter. In the proposed device, $k = 16$.

III. Top Level Architecture

The target application is a High Speed Downlink Packet Access (HSDPA) scenario with 8 antennae using QPSK. With 15 orthogonal spreading codes in use, this gives a required data rate of 3.6×10^6 symbols per second, or 57.6Mbps. The chosen clock speed of 122.88MHz 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 may be refreshed as often as every 2048 chips, which requires the channel to be preprocessed. Since the preprocessing hardware is shared with the search center generator, this function needs to be performed as fast as possible. In this $0.13\mu m$ design, it is achieved within 3000 clock cycles.

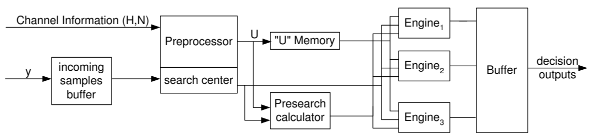


Fig. 2. Top level architecture, showing three parallel search engines interfacing to preprocessor and output logic.

Fig. 2 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 center calculations. That 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 numerical format used throughout the design is sign-magnitude floating point. The main overhead cost, compared to a fixed point system, is the need to continually rescale values to a common exponent before any addition operation, and then scale the result so that the mantissa is at least 1 and less than 2. However, a restricted floating point system was found to produce better overall efficiency of hardware use with a smaller bit width. Furthermore, it offers better accuracy and numerical stability with smaller arithmetic units [7], while the sign-magnitude format aids floating point scaling and power saving optimisations.

IV. Preprocessing

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 published 4×4 publications such as [8], and is even more challenging for an 8×8 system. Our proposed preprocessing engine is formed by the interaction of a number of smaller units, shown in Fig. 3, to perform the functions of matrix multiplication, QR decomposition, and search center calculation.

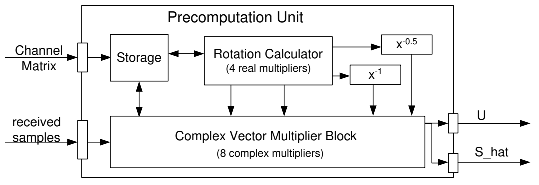


Fig. 3. Architecture of preprocessing unit for a 8x8 MIMO channel.

A. Vector-Matrix Multiplication

The majority of operations involve multiplications between pairs of matrices, or between a vector and a matrix. These are achieved by a complex multiplier array, using floating point pipelined multipliers to achieve the majority of matrix-vector calculations. Power savings are realised by only activating the registers on various paths when the data is predicted to be valid. Additionally, paths containing zero values can be predicted by checking the most significant non-sign bit on the inputs. If zero, then the result of the multiplication can be predicted as zero and the multiplication is not necessary.

B. QR Decomposition

The standard QR decomposition is very complex due to its requirement for division and square root operations. This is addressed in [9], and extended by our work in [7]. The key concept is a "scaled and decoupled" QR decomposition that separates the numerators and denominators to avoid these difficult operations. For an $m \times n$ channel matrix, $\mathbf{H}$, its QR decomposition is

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

where $\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P}$ and $\mathbf{Q}^H = \mathbf{\Phi}^H \mathbf{K}^{-\frac{1}{2}}$. The resultant $\mathbf{R}$ is an $m \times n$ upper triangular matrix, $\mathbf{Q}$ is an $m \times m$ orthonormal matrix ($\mathbf{Q}\mathbf{Q}^H = \mathbf{I}$), and such a transformation exists for any matrix.

The matrices $\mathbf{P}$ and $\mathbf{\Phi}$ are dimensioned the same as $\mathbf{R}$ and $\mathbf{Q}$ respectively, and $\mathbf{K}$ is a real-valued $n \times m$ diagonal matrix. To obtain $\mathbf{R}$ and $\mathbf{Q}$ from these results, some square roots and divisions are necessary, but we will see that these are easily addressed.

The architecture for calculating the QR decomposition breaks the algorithm into three distinct segments, as outlined in Alg. IV.1.

Algorithm IV.1 Scaled and Decoupled QR decomposition

```
For i = 1 to min(n, m) do {
    For j = i+1 to n do {
        Part 1   - Calculate Givens Rotation
        Part 2   - Recalculate $\mathbf{P}(i, j+1)$ to $\mathbf{P}(i, n)$
                 - Recalculate $\mathbf{P}(j, j+1)$ to $\mathbf{P}(j, n)$
        Part 3   - Recalculate $\mathbf{\Phi}(i, 1)$ to $\mathbf{\Phi}(i, n)$
                 - Recalculate $\mathbf{\Phi}(j, 1)$ to $\mathbf{\Phi}(j, n)$
    }
}
```

In the following, our implementation of each of these parts is discussed.

1) Givens Rotation Calculation: The "rotation calculator", shown in Fig. 4, generates a Givens matrix, $\mathbf{G}$, which is an identity matrix apart from four entries, $\mathbf{G}_{i,i}$, $\mathbf{G}_{i,j}$, $\mathbf{G}_{j,i}$ and $\mathbf{G}_{j,j}$. It also allows a new value of $\mathbf{P}(i,i)$ to be directly obtained. This part mainly consists of a series of multiplications by real numbers, and some addition and scaling operations.

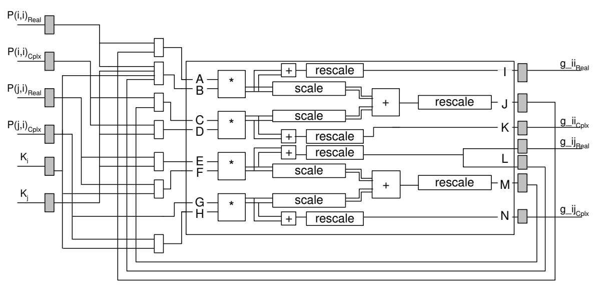
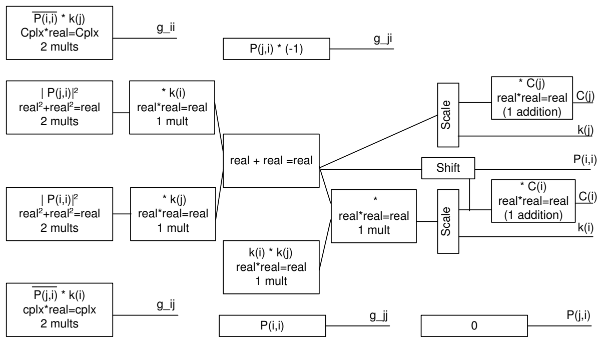


Fig. 4. Rotation Calculator Unit for QR.

The basic flow of data can be seen in Fig. 5, 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, $\mathbf{G}_{xx}$, and so these may be calculated first to allow parts 2 and 3 to proceed in parallel before this part has completed. The sign-magnitude floating point format can be very efficiently implemented, as the K and M outputs in Fig. 4 are known to be positive, and the amount of rescaling required is very limited.

Using the work timeline provided by Fig. 6, only four work cycles will be needed, regardless of the size of the decomposed matrix, and only four real number multipliers are required.



$\overline{P(i,i)}$ * k(j) Cplx*real=Cplx 2 mults
g_ii
P(j,i) * (-1)
g_ji
$| P(j,i)|^2$ real^2+real^2=real 2 mults
* k(i) real*real=real 1 mult
$| P(i,i)|^2$ real^2+real^2=real 2 mults
* k(j) real*real=real 1 mult
real + real =real
* real*real=real 1 mult
k(i) * k(j) real*real=real 1 mult
Scale
Shift
* C(j) real*real=real (1 addition)
* C(i) real*real=real (1 addition)
C(j)
k(j)
P(i,i)
C(i)
k(i)
$\overline{P(j,i)}$ * k(i) cplx*real=cplx 2 mults
g_ij
P(i,i)
g_jj
0
P(j,i)

Fig. 5. Flow of data through part 1 of the QR decomposition algorithm.

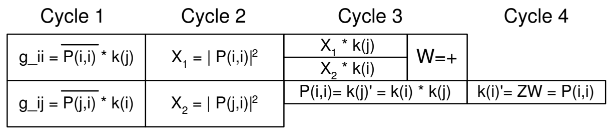


Fig. 6. Timeline of calculations for part 1 of the QR decomposition algorithm.

2) Rotation of $\mathbf{P}$*:* As seen in Fig. 7, part 2 is quite straightforward, 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 $\mathbf{P}$.

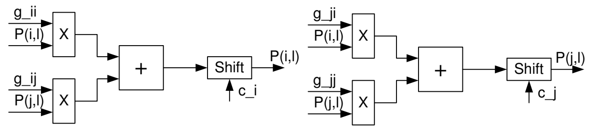


Fig. 7. Architecture for recalculation of one column of $\mathbf{P}$ or $\mathbf{\Phi}$

3) Rotation of $\mathbf{\Phi}$*:* On each loop of the decomposition, this part must multiply the existing $\mathbf{\Phi}$ by the Givens matrix $\mathbf{G}$ that was obtained from Part 1. The architecture is almost identical to that of Part 3, except that m calculations are needed for each iteration of the decomposition.

C. Search Center Calculation

While the channel decomposition only needs to be performed once per channel update, the unconstrained ML estimate, $\hat{\mathbf{s}}$, needs to be calculated for every search operation, as defined in (4). This is achieved by multiplying the incoming sample by the channel pseudoinverse, which may be expressed as :

$$\hat{\mathbf{s}} = \mathbf{P}^{-1}\mathbf{\Phi}\mathbf{y}, \qquad (7)$$

where $\mathbf{P}^{-1}$ is calculated directly. While in principle an inverse is calculated, in practice only half of the effort is required because $\mathbf{P}$ is already in an upper triangular form. Direct inversions are often avoided due to concerns over numerical stability, however in [7] we demonstrated that it is no different to the commonly recommended back-substitution method.

D. Scalar Reciprocal and Reciprocal Square Root

While the scaled and decoupled method greatly reduces the number of square roots and divisions required, it does not completely eliminate them. In particular, to obtain $\mathbf{R}$, the calculation is

$$\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P}. \tag{8}$$

The inversion and reciprocal square root operations are implemented by simple application of the bisection algorithm. The bisection algorithm assumes the known existence of a solution between two points, a and b. On each iteration, a midpoint $c = \frac{a+b}{2}$ is selected, and it is decided whether the 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 :

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

Conveniently, the computational load is quite small when applied to the binary domain. For the reciprocal square root in a floating point system, the original value is $x_m 2^{x_e}$, so the result will be of the form $y_m 2^{\lfloor \frac{x_e}{2} \rfloor}$. In a floating point system, it is known that x_m is between $1.00\ldots00_2$ and $1.11\ldots11_2$. Therefore, y_m will be between $0.10\ldots00_2$ and $1.00\ldots00_2$ for both the inverse and inverse square root operations.

The operation is executed by iterating once for each bit of the result. The test case C is the progressive result with a 1 added to the left as a new least significant bit. If the cost of C is positive, then the decision is kept, otherwise that bit is set to zero and the previous decision is kept.

By examination of the operations involved, it can be seen that only a series of shifts and additions are required. This allows for the simple architectures shown in Fig. 8 and Fig. 9.

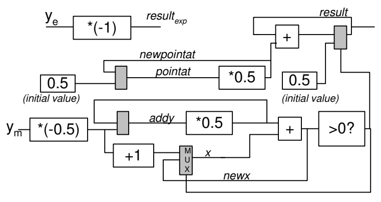


Fig. 8. Scalar Inverse Unit. Shaded boxes represent flip-flops.

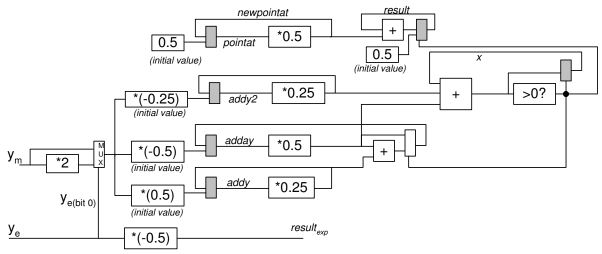


Fig. 9. Inverse Square Root Unit. Shaded boxes represent flip-flops.

E. Implementation

A bit-accurate software model of the preprocessor reveals the effect of varying the manitissa bit size, and indicates (Fig. 10) that as little as 10 mantissa bits are necessary to obtain near-optimal performance. The final design implements a 12-bit version with 20000 standard cells and 9216 bits of memory.

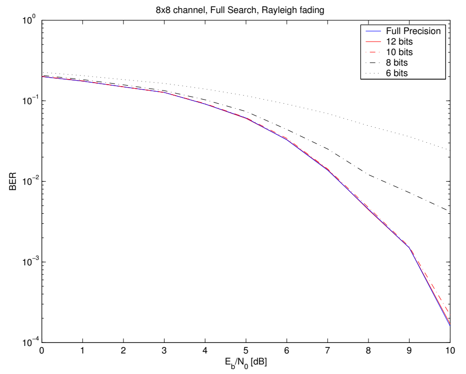


Fig. 10. Performance of preprocessor under various bit widths for an 8x8 MIMO channel.

V. Search Engines and Presearch

The search engines are designed for expanding the 64 children of 16 candidate nodes and performing a sort, but the 16 candidates do not exist until after the second level of a quaternary tree. Hence, no sort is required and less calculations are required to fully expand the nodes that are present.

To exploit this, a presearch unit is implemented and optimised to calculate the first two stages of the search in an efficient manner. This is done as fast as the search center estimates are made available, and so does not negatively impact on the performance.

The search engines 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. 11, and implements a series of simple node cost calculations in floating point format to evaluate (5) and expand each successive level of the tree. Once these are calculated, a sorter block is used to determine the best 16 of the 64 generated options.

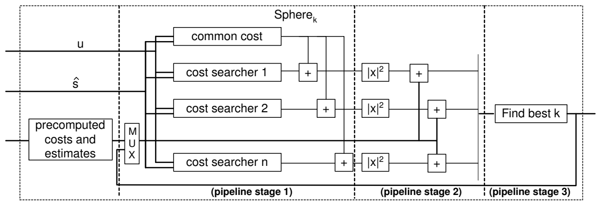


Fig. 11. Architecture of a single search engine.

By including the presearch unit, two iterations are removed from the engines and this increases their throughput by 33%, reducing the total number of engines required.

VI. 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 122.88MHz. 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 [10], optimised for this application, was chosen.

Fig. 12 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 2^2 inputs, the third sorts 2^3, and so on. Within each stage, a series of pairwise comparisons are made to gradually reorder the elements in a set of substages.

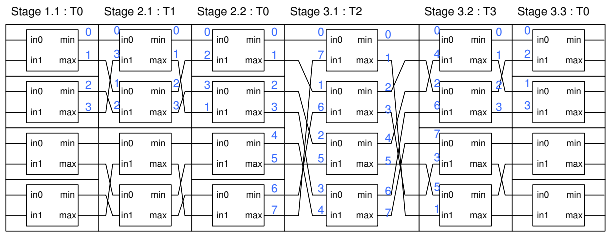


Fig. 12. 8 input sorting network configuration. The numbers indicate the matching inputs and outputs of connecting stages for clarity.

Fig. 12 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 $2^n - k - 1$.
- 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, 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. 13.

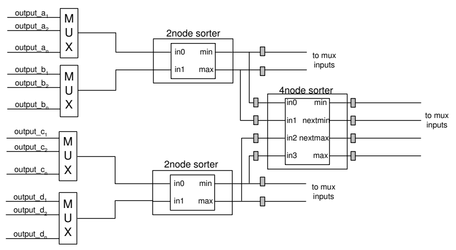


Fig. 13. Optimised structure of the sorting elements. The 2-node elements implement the majority of the stages, using multiplexers to switch the inputs, and the 4-node unit implements the combined T3/T0 stage. The grey boxes represent registers separating the stages.

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.

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

VIII. Results and Comparison

A prototype device has been prepared and routed (Fig. 14) for a 0.13μm technology, with a core size of approximately 12mm^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, presented in [11], [5], indicates near-ML results.

Reference	Proposed	[8]	[2]	[1]
Preprocessor	20K Gates	No	No	ML-APP
Antennae	8×8	4×4	4×4	4×4
List decoder	Yes	No	Yes	Yes
Constellation	QPSK	16QAM	16QAM	QPSK
Throughput, 1dB	$>$ 57.6Mbps+	$\ll$ 50Mbps	106Mbps	28Mbps
Throughput, 5dB	$>$ 57.6Mbps+	$<$ 50Mbps	106Mbps	28Mbps
Gates	500K	117K	97K	170K
Clock Speed	122MHz	51MHz	200MHz	122MHz

TABLE I
COMPARISON OF MIMO IMPLEMENTATION PROPOSALS

Table I 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 and [8] does not provide soft decisions. 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.

Ref [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.

IX. CONCLUSION

This paper has presented a unique scalable device based on a k-best algorithm, establishing the feasibility of decoding data from an 8×8 MIMO system at 56Mbps for all SNRs. 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.

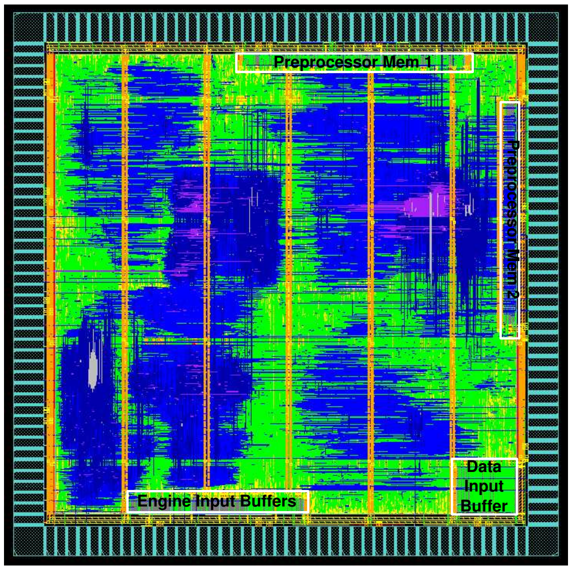


Fig. 14. Plot of the final chip, indicating the relative sizes of memories and core logic.

REFERENCES

[1] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, and G. Knagge, "A 28.8 Mb/s 4x4 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, Feb 2004.

[2] Z. Guo and P. Nilsson, "Algorithm and implementation of the k-best sphere decoding for MIMO detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 491–503, March 2006.

[3] L. M. Davis, D. C. Garrett, G. K. Woodward, M. A. Bickerstaff, and F. J. Mullany, "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, Apr 2003.

[4] B. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Information Theory*, vol. 51, pp. 389–399, Mar 2003.

[5] G. Knagge, G. Woodward, B. Ninness, and S. Weller, "An optimised parallel tree search for multiuser detection with VLSI implementation strategy," in *Global Telecommunications Conference (IEEE GLOBECOM)*, pp. 2440–2444, Dec 2004.

[6] S. Verdú, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *IEEE Transactions on Information Theory*, vol. 32, pp. 85–96, Jan. 1986.

[7] G. Knagge, L. Davis, G. Woodward, and S. R. Weller, "VLSI preprocessing techniques for MUD and MIMO sphere detection," in *Australian Communications Theory Workshop (AusCTW) 2005*, pp. 204–210, Feb 2005.

[8] A. Burg, M. B. amd M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid State Circuits*, vol. 40, pp. 1566–77, July 2005.

[9] L. M. Davis, "Scaled and decoupled cholesky and QR decompositions with application to spherical MIMO detection," *IEEE Wireless Communications & Networking Conference (WCNC)*, pp. 326–331, Mar 2003.

[10] N. K. Sharma, "Modular design of a large sorting network," in *Third International Symposium on Parallel Architectures, Algorithms, and Networks, 1997. (I-SPAN '97).*, pp. 362–8, Dec 1997.

[11] G. Knagge, G. Woodward, S. R. Weller, and B. Ninness, "A VLSI optimised parallel tree search for MIMO," in *Australian Communications Theory Workshop (AusCTW) 2005*, pp. 198–203, Feb 2005.

[12] A. Burg, M. Borgmann, M. Wenk, C. Studer, and H. Bolcskei, "Advanced receiver algorithms for mimo wireless communications," *Proc. Design Automation and Test in Europe Conf. (DATE)*, March 2006.