Matrix Datapath Architecture for an Iterative 4x4 MIMO Noise Whitening Algorithm

Geoff Knagge
University of Newcastle
68 Waterloo Road
North Ryde, NSW 2113
Australia
knagge@lucent.com

David Garrett
Sivarama Venkatesan[*]
Chris Nicol
Bell Labs Research (Lucent Technologies)
68 Waterloo Road
North Ryde, NSW 2113
Australia
garrettd,sivarama,chrisn@lucent.com

ABSTRACT

This paper describes a power, speed and area efficient VLSI implementation of a noise whitening algorithm for a 4x4 MIMO channel. The architecture combines innovative use of Hermitian matrices to streamline the iterative calculations, with a 4x1 matrix row-column multiplier as the core component. The optimisations in the datapath reduce the power and latency needed to implement the algorithm. The Booth recoded complex multipliers use logic sharing to reduce power and complexity, and incorporate low-power sleep logic that does not increase the critical path. The design has been successfully synthesised in a 0.18μm, 1.8V CMOS technology, and has the potential to be adapted to other applications requiring matrix multiplication.

Categories and Subject Descriptors

B.6 [**Logic Design**]: Design Styles; C.1 [**Processor Architectures**]

General Terms

Design

Keywords

Noise Whitening,MIMO,Booth recoding,matrix multipliction

1. INTRODUCTION

Multiple-input multiple-output (MIMO) techniques are of significant research interest due to their capability to dramatically increase the data rate achievable over a wireless channel without requiring additional transmit power or bandwidth [1, 2].

A block diagram of a detector for a MIMO system with M transmit antennas and N receive antennas is shown in figure 1. When the channel is highly frequency-selective, a linear space-time equaliser may be deployed at the front end of the receiver, in order to mitigate the spatial and temporal self-interference introduced by the channel. In order to obtain even better spatial separation of the transmitted signals, it may further be necessary to pass the output of the space-time equaliser through a maximum-likelihood (ML) detector. It is usually convenient to build the ML detector under the assumption that the noise at its input is spatially white. However, the equaliser preceding the ML detector colours the noise at the receiver input. Ignoring the colouring of the noise at the input of the ML detector can result in significantly degraded performance. It is therefore important to have a "noise whitening" stage between the equaliser and the ML detector, as shown in figure 1.

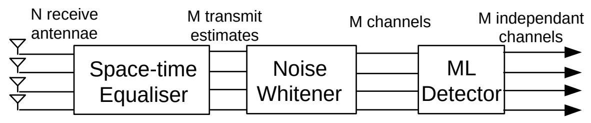

Figure 1: The proposed noise whitener attempts to counteract the filtering effect imposed on the AWGN channel noise

The standard method of noise whitening is based on the Cholesky decomposition of the covariance of the noise at the output of the equaliser. However, this method requires arithmetic divisions and square roots, operations that are preferable to avoid in hardware due to their complexity.

To circumvent the need for divisions and square roots, an iterative algorithm based on Newton's method has recently been devised to perform the noise whitening [3]. This paper describes an efficient hardware architecture for implementation of this algorithm, and presents an indication of the power, speed and area parameters for implementation in a 4x4 MIMO detector.

The rest of the paper is organised as follows. Section 2 introduces the algorithm used to solve the noise whitening

[*]Based at Bell Labs Research, NJ, USA.

problem. Since the algorithm is heavily based on multiplication, section 3 describes our design of a power and speed efficient multiplier circuit with sleep capabilities. However, we are concerned with the multiplication of matrices, and section 4 considers various architectures before describing our chosen technique and optimisations used in its implementation. Section 5 reveals how our noise whitener implements this, and section 6 provides results that indicate the speed, power, and area parameters of its implementation.

2. NOISE WHITENING ALGORITHM

The algorithm uses an iterative process to determine the noise whitening matrix, which is then used to filter the vectors of values being output from the equaliser. The key feature of this algorithm that provides scope for innovation in its implementation is that it uses Hermitian matrices. That is, most of the data is manipulated in matrices that are the conjugate transpose of themselves.

The spatial covariance of the noise at the ouput of the equaliser can be expressed as $\mathbf{K} = \mathbf{W}\mathbf{W}^H$, where $\mathbf{W}$ is a matrix of equaliser coefficients. The noise whitening problem is then to find a matrix $\mathbf{Q}$ such that $\mathbf{QKQ}^H = \mathbf{I}$, the identity matrix. Such a $\mathbf{Q}$ is guaranteed to exist because $\mathbf{K}$ is necessarily Hermitian and positive-definite.

In the system of interest, the number of transmit and receive antennas were both equal to 4, and the size of the $\mathbf{W}$ matrix was 4x128, so that the $\mathbf{K}$ matrix and the desired $\mathbf{Q}$ matrix are both 4x4.

The implemented iterative noise whitening algorithm is

$$\mathbf{Q_{n+1}} = \frac{\mathbf{Q_n}}{2} * (3\mathbf{I} - \mathbf{K}\mathbf{Q_n}^2)$$

To ensure convergence, the iterations start with $\mathbf{Q_0} = \alpha\mathbf{I}$, where α is any positive real number smaller than $\sqrt{\frac{1}{trace(\mathbf{K})}}$. It is easy to see that all the iterates $\mathbf{Q}_n$ are Hermitian matrices, since $\mathbf{Q}_0$ and $\mathbf{K}$ are themselves Hermitian.

After the final $\mathbf{Q_n}$ is calculated after several iterations, the noise whitening filtering operation can be done, by multiplying $\mathbf{Q_n}$ by the non-whitened 4x1 matrix sample, $\mathbf{s}$.

3. MULTIPLIER DESIGN

It is critical to have a complex number multiplier that is size and power efficient, while meeting timing requirements. The multiplier used was based on radix-4 Booth recoding and carry-save arithmetic, and optimised for speed and power with a unique sleep logic technique. The multiplier also uses standand sign extension avoidance techniques [4], which can be optimised for complex number multipliers.

3.1 Sleep Logic

The provision of a sleep function, to disable multipliers when they are not required, can considerably reduce power consumption. An obvious method is to simply gate the inputs, but this will increase its critical path delay.

Our solution is found by examining the Booth recoding logic. The booth recoder generates three signals, X2, X0, and NEG to tell the partial product generator to shift left, zero the partial product, or take its complement. The logic for these signals is seen in figure 2 and, making the reasonably accurate assumption that each gate has a similar propagation delay, it can be seen that all signals are generated simultaneously.

Figure 2 also shows the generation of the least significant partial product bit, from which it can be seen that the enable function is achieved by simply gating the Booth recoded “X0” signal. It can also be seen that the critical path delay does not depend on this gate, and so a sleep function has been achieved without increasing that delay.

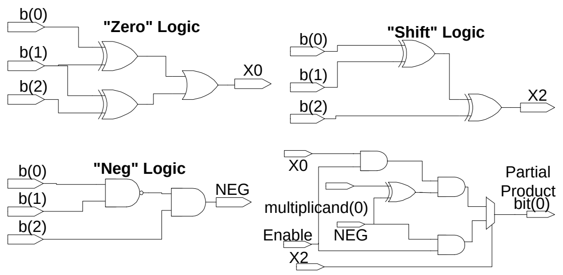

Figure 2: Achievement of a multiplier sleep function without increasing the critical path

3.2 Complex Multiplier

Reductions in gate counts, speed, and power usage for the complex multipliers have been achieved through sharing of common logic in the multipliers. Although a complex multiplier is made of four real number multipliers, only two booth recoders are required. In addition, much of the error correction terms are constant, so it is possible to precompute the sum of corresponding pairs of these and replace them with a single term. By modifying the Booth recoding, our multipliers multipliy by the conjugate of one of its inputs, for reasons that are described in section 4.2.

It is theoretically possible to implement complex multipliers with only three real number multipliers, such as those described by [5], but these require use of carry propagate additions or lookup tables, and provide little benefit for this synthesisable design.

4. MATRIX MULTIPLICATION

The algorithm consists of two types of operation, being the calculation of the whitening matrix for the current estimation of the channel, $\mathbf{Q_{n+1}} = \frac{\mathbf{Q_n}}{2} * (3\mathbf{I} - \mathbf{K}\mathbf{Q_n}^2)$, and the final filtering operation $result = \mathbf{Q_n s}$. The first can be broken down into four calculations, or modes:

$$\mathbf{Q_{n+1}} = \underbrace{\frac{\mathbf{Q_n}}{2}(3\mathbf{I} - \overbrace{\overbrace{\mathbf{W}\mathbf{W}^H}^{mode00}\ \overbrace{\mathbf{Q}_n^2}^{Mode01}}^{mode10}\ )}_{mode11} \qquad (1)$$

This shows that we have a total of three multiplications of 4x4 matrices to perform on each iteration. The result of the calculation of $\mathbf{W}\mathbf{W}^H$ is constant for each iteration, and so can be precomputed as the matrix $\mathbf{K}$.

4.1 Multiplication Tradeoffs

The principle operation of interest is the multiplication of pairs of 4x4 matrices, and so it was necessary to determine which of the range of possible matrix multiplication architectures was most suitable to this application. In addition to minimising the circuit area used, the design needed to work within a set time budget. Each output cell of the resultant matrix is the multiplication of a corresponding row and column from the inputs, requiring four multiplications.

At one extreme, a fully parallel approach could execute in one clock cycle with 64 complex multipliers. Assuming a synthesised 16bit complex multiplier uses around $0.09mm^2$, this would use $5.8mm^2$ of silicon alone. Since it is economically desirable to reduce the area, this is not practical.

The other extreme is to only instantiate a single multiplier, and perform all calculations sequentially. To calculate an entire matrix, we would require $4 * 16 = 64$ operations (clock cycles). With ten iterations of three multiplications per iteration, plus the calculation of $\mathbf{K}$, this is around 4000 clock cycles to calculate the whitening matrix. For this application, that is too slow.

It is therefore obvious that a compromise is required between excessive circuitry and excessive latency. The hybrid approach considered here achieves this, and is effectively a vector multiplier that is successively used sixteen times, with only four multipliers, to produce the output matrix. By using optimisations based on the presence of Hermitian matrices, both the time and power usage of this architecture can be further reduced.

The trade-offs between architectures can be seen in tables 1 and 2. The tables assume that the algorithm runs for 10 iterations, and that all necessary data is instantly available to the parallel architectures. The non-parallel values make the assumption, explained in secion 5.2, that two cycles are needed to obtain values for the calculation of non-diagonal outputs of K.

architecture	complex multipliers	cycles per matrix	cycle per algorithm
Full Parallel	64 (2048)	1 (1)	31
Parallel	64	1 (32)	62
Sequential	1	64/128 (3584)	11264
Hybrid	4	16/32 (896)	1536

Table 1: Characteristics of various matrix multiplication architectures, with values in parentheses indicating calculation of $K = WW^H$.

architecture	complex multipliers	cycles per matrix	cycle per algorithm
Full Parallel	40 (1280)	1 (1)	31
Parallel	40	1 (20)	50
Sequential	1	40/80 (2048)	3648
Hybrid	***4***	***16 (448)***	***928***

Table 2: Benefits of using Hermitian optimisations

4.2 Memory optimisations

In order to realise the full speed potential of the chosen architecture, we need to obtain four input values in one clock cycle. To make this possible, matrices are stored with entire rows in each memory location, with locations partitioned to allow individual matrix cells to be written with respective write enable signals (Table 3). The output addresses are thus partitioned so that the two least significant bits determine the partition number, and the most significant bits are the physical address.

Address	Bits 31-0	63-32	95-64	127-96
00	$A(1,1)$	$A(1,2)$	$A(1,3)$	$A(1,4)$
01	$A(1,2)^*$	$A(2,2)$	$A(2,3)$	$A(2,4)$
10	$A(1,3)^*$	$A(2,3)^*$	$A(3,3)$	$A(3,4)$
11	$A(1,4)^*$	$A(2,4)^*$	$A(3,4)^*$	$A(4,4)$

Table 3: Memory locations can be partitioned so that an entire matrix row of the Hermitian matrix may be read in one operation.

To read a given row of the matrix, we now only need to access the corresponding memory address. By realising that the values each column of a Hermitian matrix is the conjugate of a corresponding row, all that is needed to multiply by a column of the matrix is to read its corresponding row, and multiply by the conjugate of the values read. Since the multipliers were designed to multiply by the conjugate of one input, this is efficiently achieved.

To generate a Hermitian matrix, it is only necessary to calculate the upper triangle by multiplication, since the lower triangle is simply its conjugate and can be formed by negation of the complex components. For a NxN matrix, this means only $\sum_{n=1}^{N} n$ calculations are needed rather than N^2. This technique saves significant power, since the multiplication is the most power consuming part of the circuit. With this technique, the output matrix is written in the order depicted by figure 3.

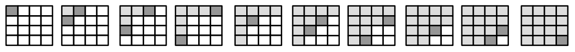

Figure 3: Order in which the Hermitian matrices are filled in. Dark squares represent the cells currently being written, and light squares represent the cells already calculated.

5. PROCESSOR ARCHITECTURE

With an efficient implementation stategy in place for the noise whitener, a five stage pipeline can then developed. These stages consist of address generation, a memory wait, decoding of memory data, multiplication, and finalisation of results. In this final stage, if the calculation being performed is $\mathbf{K} = \mathbf{WW}^H$, then the results are tallied until the entire output cell has been calculated. When that is the case, and for all other calculations, the result is resolved with a carry-propagate adder and written to the memories.

5.1 State and address generation

The state of the whitener is simply a record of which of the five calculations is being performed, consisting of the two types of calculation and each of the modes shown in equation 1.

The monitoring of our progress in a particular state is done through incrementing the output address as is appropriate for our Hermitian output optimisations.

For the 4x4 matrices used here, the output address is simply the concatenation of a column number and a row number. Since the output is also Hermitian, we only need to generate the addresses of the upper triangle and allow the output stage to handle the other half of the output. In addition, with the technique used to store data in memories, no additional logic is required to generate the read addresses because these can be directly derrived from the row and column number that form the output address. Mode 00, however, does require an additional counter to address the W memory.

Since the output stage needs to perform two write cycles for non-diagonal entries, it is necessary to introduce a pipeline stall for those entries. Such a stall is not necessary for diagonal entries, and this can be detected when the row and column numbers are equal. We save some clock cycles with this optimisation in the calculation of $\mathbf{K} = \mathbf{W}\mathbf{W}^H$, but the main benefit is that we save power because we do not need to recalculate essentially the same values for the lower triangle of the output matrix.

5.2 Mode 00: $\mathbf{K} = \mathbf{W}\mathbf{W}^H$

This calculation is the first that is done in the noise whitening initialisation, and we assume that $\mathbf{W}$ is preloaded. This state is also where $\mathbf{Q_0}$ is initialised as a diagonal matrix with the provided real value α.

We constrain the width of matrix $\mathbf{W}$ to be a multiple of 4, so that it can fit neatly into the architecture. Since the one matrix is used as both inputs for this calculation, two reads are generally needed to obtain the required row and column, remembering that the "column" is actually the conjugate of the respective row. For the diagonals, only one read is needed since the row and column are the same.

5.3 The filtering operation

This state is entered once the final $\mathbf{Q_n}$ has been calculated and is the same as any other, expect that the output is written in four clock cycles to flip-flop outputs instead of a memory, and is a four cell column vector.

5.4 Output of results

Since the architecture is optimised to minimise the amount of power consuming calculations required, the output needs to write to the upper and lower triangle of the Hermitian result. This stage is also where the division by two and the "$3\mathbf{I} - \mathbf{x}$" operation in carry-save form is done if necessary, before being resolved with a carry propagate adder.

The Hermitian optimisation is a simple matter of saving the results to a temporary register if the generated value is not on a diagonal. On the next clock cycle, we can then take the conjugate of the saved values, and write it to the lower triangle by swapping the least and most significant halves of the address. The pipeline stall introduced in the Read stage will allow time for this to be done

6. HARDWARE PERFORMANCE

To demonstrate the feasibility of the algorithm's implementation, and to show the benefits of the optimisations described, the architecture was implemented with VHDL and synthesised, with varying amounts of numerical precision, using a 0.18μm, CMOS process in the slow corner. The target clock cycle of 8.2ns used is simply a value of interest, as the 16 bit design has been shown to be synthesisable to a clock cycle of under 6.5ns.

The results are in table 4, where power is averaged from the loading of the W matrix until the last filtering operation. An indication of these power savings is easily gained by considering the "maximum power" column power for each configuration. The maximum power is the power dissipated during the calculation of $\mathbf{K} = \mathbf{W}\mathbf{W}^H$, where the Hermitian optimisations are of little significance. This uses over 100mW more than the rest of the iterative process, proving the significant benefit of this technique.

The actual configuration used in any given application will be a compromise between performance and size and power. As table 4 shows, a lower numerical precision gives a smaller and less power consuming circuit, but the performance of the algorithm is degraded.

Bits	Size	Average Power	Maximum Power
13	$1.01mm^2$	119 mW	240 mW
14	$1.07mm^2$	129 mW	260 mW
15	$1.20mm^2$	159 mW	306 mW
16	$1.27mm^2$	193 mW	341 mW

Table 4: Comparison of size and average power for different amounts of numerical precision.

7. CONCLUSION

This paper has described the successful implementation of a noise whitening algorithm that is targeted to MIMO applications and does not require VLSI-unfriendly operations such as division. The major components, the multipliers, have been optimised and designed with sleep logic, saving a considerable amount of power. Architectural optimisations exploit the Hermitian nature of the matrices used for multiplication, and save further power as well as speed and area.

While this architecture was designed specifically for noise whitening of MIMO channels, it has the potential to be adapted to many other MIMO algorithms requiring the use of matrix multiplication. In fact, its application need not restricted to MIMO, as noise whitening and matrix multiplication in general is required in numerous other applications. Nor is it restricted to multiplication of 4x4 matrices, or square matrices. Multiplication of rectangular matrices can be handled in a similar manner to the calculation of $\mathbf{K}$. The architecture is easily scalable to larger or smaller matrices as long as the physical size and number of multipliers remains practical.

8. REFERENCES

[1] G. J. Foschini and M. J. Gans, "On limits of wireless communications in a fading environment when using multiple antennas," *Wireless Personal Communications*, vol. 6, pp. 311–335, March 1998.

[2] I. E. Telatar, "Capacity of multi-antenna gaussian channels," *European Trans. Telecomm.*, vol. 10, pp. 585–596, November-December 1999.

[3] S. Venkatesan, L. Mailaender, and J. Salz, "Iterative algorithms for noise whitening." Bell Labs Technical Memorandum, in preparation.

[4] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045–1047, December 1973.

[5] Y. B. Mahdy, S. A. Ali, and K. M. Shaaban, "Algorithm and two efficient implementations for complex multiplier," in *Proceedings of ICECS*, pp. 949–953, 1999.