Adaptive MIMO Decoder

EE290C Project Final Report

Antonio Puglielli and Simon Scott

I. Objectives

MIMO refers to the use of multiple transmit antennas *and* multiple receive antennas to spatially multiplex data streams over a channel. This channel is often represented by a complex matrix (H), and can change over time. Although MIMO can be used to increase reliability of a link, this project is only concerned with the use of MIMO to improve throughput. Furthermore, only the single user case will be considered, meaning that all the receive antennas are assumed to be on the same hardware and can share information.

The objectives of this project are therefore to develop a VLSI implementation of a MIMO decoder that:

- supports the antenna configurations and modulation schemes used in both 802.11ac and LTE.
- meets both the 802.11ac and LTE throughput requirements.
- is able to adapt to changing channel conditions.
- is both compile-time and run-time configurable.

II. Algorithm Evaluation and Selection

The two main classes of MIMO detection algorithms are those of linear and nonlinear algorithms. Linear detection algorithms refer to techniques which form the detected symbols as a linear combination of the received data. Nonlinear detection schemes are more complex, but usually achieve better performance.

The maximum likelihood (ML) receiver is a nonlinear scheme which is theoretically optimal. Based on the received data and the channel estimate, it computes the most likely transmitted vector. However, its complexity scales exponentially with the MIMO order since the number of different transmit sequences increases exponentially with the number of spatial streams. Sphere decoding is a simplification of this which achieves cubic complexity by restricting the search space to a sphere of finite radius.

On the other hand, linear schemes require lower computational complexity but usually entail a loss in performance. These detection methods attempt to separate out the transmitted streams by linearly filtering the received vector. In zero-forcing (ZF) detection, the receive filter is chosen to minimize inter-stream interference. Though this technique performs well in interference-limited scenarios, there is a substantial loss in performance when noise dominates. Therefore, it is better to use the minimum mean square error (MMSE) receive filter since it minimizes the total mean square error in the detected vector. In the high SNR limit, this converges to the ZF receiver. Finally, the MMSE receiver can be combined with successive interference cancellation (SIC), where each spatial stream is successively decoded and its contribution subtracted from the received data. Therefore, streams decoded later in the process see a diminished amount of interference. This technique also theoretically achieves the full capacity, though in practice its performance is reduced compared to ML because it is more sensitive to errors in decoding the initial streams [1].

An additional metric of interest is the ability to track channel variations. Since real channels vary between training sequences, it is desirable for the MIMO receiver to track these channel changes and maintain good performance. It is little use to invest in complex and power-hungry receiver hardware if this hardware cannot track the channel state and therefore suffers a large performance degradation. Consequently, it is of interest to add the ability to adapt on top of one of these MIMO detection schemes. This is easiest to accomplish with a linear detection scheme since adaptation can be implemented simply by modifying the receive filter, easily realized using a well-known technique such as least mean squares (LMS) or recursive least squares (RLS) [2], [3]. When the adaptive process is seeded with the initial MMSE receive filter, the convergence is immediate and tracking ability is improved.

To evaluate the performance of these algorithms, we simulated the uncoded symbol error rate (SER) versus SNR curves for a 4x4 MIMO system. Figure 1 shows the performance achieved by the different receive techniques in a static 4x4 channel realized using the 802.11n channel model B. As expected, the ML and sphere decoding achieve the best performance, and unseeded LMS (which uses a very long training sequence) the worst, while seeded (MMSE) LMS and direct processing (MMSE) achieve very similar performance. Figure 2 shows the same simulation conducted with a time-varying 802.11n model B channel using a maximum doppler shift of 3 Hz. Here we can see that all receive schemes suffer a substantial performance loss, but that seeded LMS achieves a SER comparable to that of ML for high SNRs due to its ability to track the channel variations.

Adaptation can be combined with the ./bui MMSE linear detection scheme into the following algorithm:

1) Estimate the channel matrix based on *a priori* known training sequence.
2) Compute initial MMSE receive filter
3) Begin processing received samples using receive filter matrix.
4) Adapt receive filter using LMS algorithm to track channel variations.

We implemented an LMS-MMSE MIMO decoder since this has relatively low implementation complexity but can show

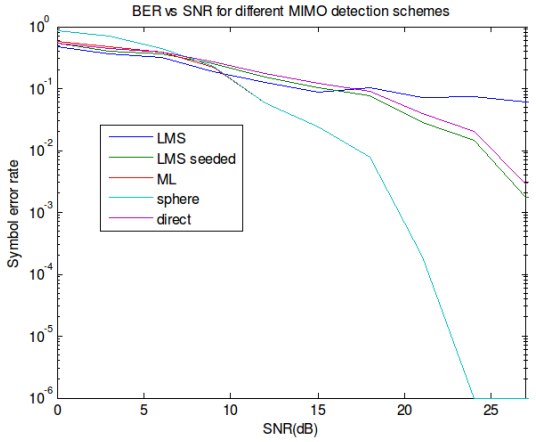

Fig. 1. SER vs SNR for different MIMO detection schemes with a static channel

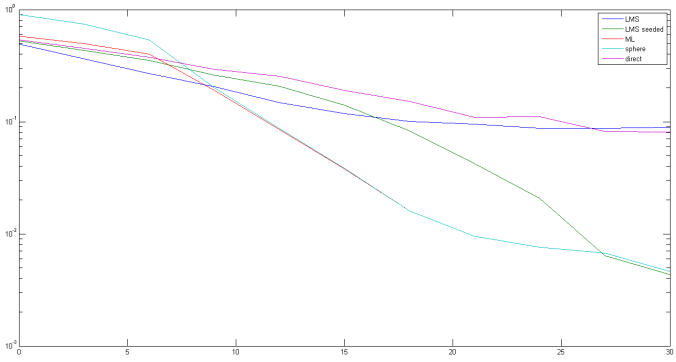

Fig. 2. SER vs SNR for different MIMO detection schemes with a time varying channel (doppler shift of 3Hz)

the usefulness of this technique and is easily extendable to more complicated receive processing. For example, MMSE-SIC detection can be accomplished by adaptively estimating the channel and performing the decoding one stream at a time instead of simultaneously.

III. Hardware Architecture

The architecture of the Adaptive MIMO Decoder is shown in Figure 3. The host configures the decoder by writing to the configuration registers, setting parameters such as the number of antennas and the modulation scheme. Since different standards use different training sequences, the host then needs to write the current training sequence to the training memory. Once complete, the host writes a 1 to the *start* register, and begins sending samples to the RX Data Queue.

The Channel Estimator uses the training sequence and the first few received samples to estimate the channel matrix H. This channel matrix is then sent to the Initialize Weights module, which computes the initial decoder matrix W_seed. The Adaptive Decoder uses this W_seed as a starting point for its decoder matrix W. The Adaptive Decoder decodes the samples in the RX Data Queue and writes the decoded symbols to the Decoded Data Queue. For each set of samples, the Adaptive Decoder also updates the W matrix.

Finally, the host reads the decoded symbols from the Decoded Data Queue. Each of the different modules in the design will now be described in more detail below.

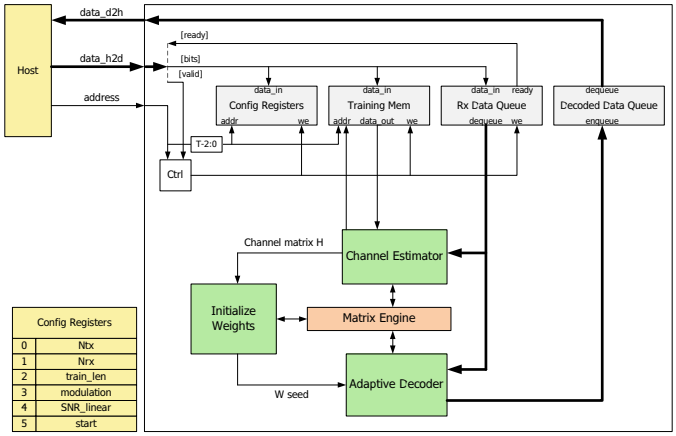

Fig. 3. The hardware architecture of the LMS Adaptive MIMO Decoder

A. Matrix Engine

The Matrix Engine computes the complex vector-matrix multiply between an input vector V and an input matrix M. The input matrix M has size $N_{transmit\_antennas} \times N_{receive\_antennas}$. Since the Adaptive Decoder needs to decode a new set of antenna samples every cycle, the Matrix Engine needs to compute the vector-matrix multiply in a single cycle. For the 4x4 MIMO case, it does this using four 4-element dot-product units which operate in parallel. However, if only 2x2 MIMO needs to be supported, then only two dot-product units will be created at compile time.

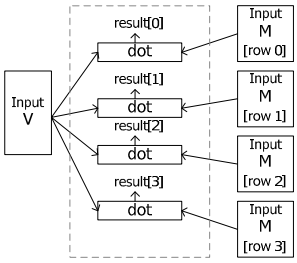

Fig. 4. Design of the matrix engine

B. Channel Estimator Module

The channel estimator module uses the known training sequence as well as the corresponding received symbols to compute the initial channel estimate. We have used a training sequence which is orthogonal in space and time, mimicking the general format of the 802.11ac VHT-LTF fields but not the specifics. Since the training sequence is an orthogonal matrix, the channel estimate can be obtained simply from a matrix multiplication of the training sequence matrix with the received signals matrix. The channel estimator module uses the matrix engine to perform this computation and then sends the result to the weight initialization module.

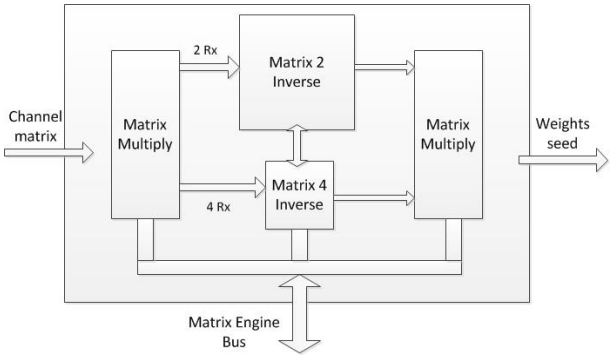

Fig. 5. Weight initialization module

C. Initialize Weights Module

The initialize weigts module computes the MMSE receive filter using the initial channel estimate, H:

$$W = (H^H H + \sigma^2 I)^{-1} H^H$$

The main operations required are two matrix multiplications and one matrix inversion. The matrix multiplications are performed using the matrix engine, while a specialized 2x2 matrix inversion hardware is contained within this module. For 2x2 MIMO cases it is sufficient to simply use this module, while for larger MIMO orders the matrix is inverted blockwise using the 2x2 matrix inversion engine and small additional hardware.

1) 2x2 Matrix Inversion: For 2x2 matrices, the inverse can be computed efficiently using the explicit formula:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}^{-1} = \frac{1}{x_{11}x_{22} - x_{12}x_{21}} \begin{bmatrix} x_{22} & -x_{12} \\ -x_{21} & x_{11} \end{bmatrix}$$

This calls for two complex multiplications and a subtraction (to compute the determinant) and four complex divisions. Since the complex division requires a large and power-hungry hardware, we use a single complex divider (pipelined in three stages) which is multiplexed to perform the four divisions per 2x2 matrix. Thus, inversion of a 2x2 matrix takes 13 cycles (including one for determinant computation).

2) Inversion of Larger Matrices: To invert larger matrices, the 2x2 matrix inversion hardware and the matrix engine are used. The inversion is performed blockwise:

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

$$\begin{bmatrix} -A^{-1}B(D - CA^{-1}B)^{-1} \\ (D - CA^{-1}B)^{-1} \end{bmatrix}$$

NEED TO FIX EQUATION TO USE MULTILNE

For a 4x4 matrix inversion, this requires two 2x2 matrix inversions as well as several 2x2 matrix multiplications, which are performed using their respective dedicated hardware. We currently have not implemented 3x3 matrix inversion, but it can accomplished with minimal hardware overhead by using this same approach.

D. Adaptive Decoder Module

The task of the Adaptive Decoder module is to decode the samples (received from the antennas) into the symbols for the independent data streams. It does this by multiplying the received samples (vector x) with the decoder matrix W, using the Matrix Engine. This is shown in Figure 6. The resulting Wx vector is then converted into a vector of symbols y using a simple slicer to find the nearest constellation points. The $error$ between Wx and the decoded symbols is computed and multiplied by a step-size parameter μ.

This weighted error is used to update the decoder matrix according to the following equation:

$$W_{next} = W - x \times (error \cdot \mu)$$

Note that $x \times (error \cdot \mu)$ is the vector cross-product between received samples x and the error weighted by μ.

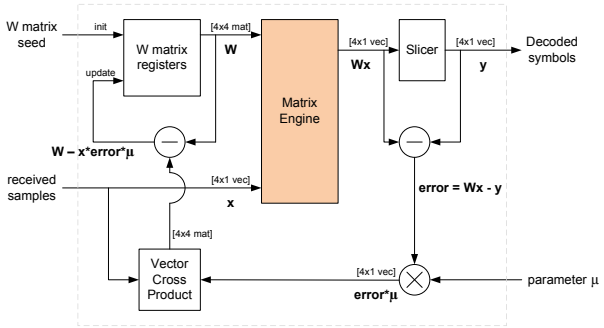

Fig. 6. The adaptive decoder module

E. Evaluation of Bitwidths

The hardware architecture described above was implemented in Chisel [4]. In order to verify correctness and determine the optimum bitwidths to use, the Chisel implementation was tested using test vectors taken from the MATLAB simulations (as described in Section II). The symbol error rate versus SNR plots for different bitwidths, using a static channel, are shown in Figure 7. This plot also includes the results from the MATLAB simulation for comparison. Similarly, Figure 8 shows the SER vs SNR plots for a slowly changing channel.

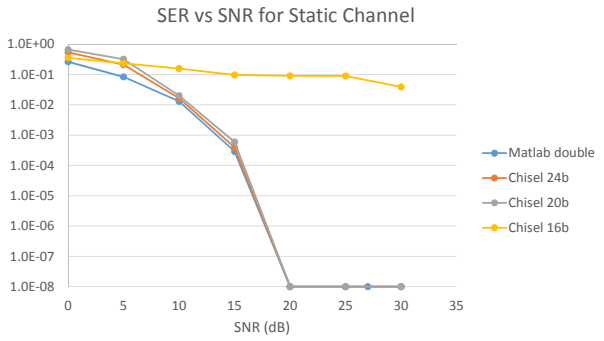

Fig. 7. SER vs SNR curves for Chisel implementations with different bitwidths (static channel)

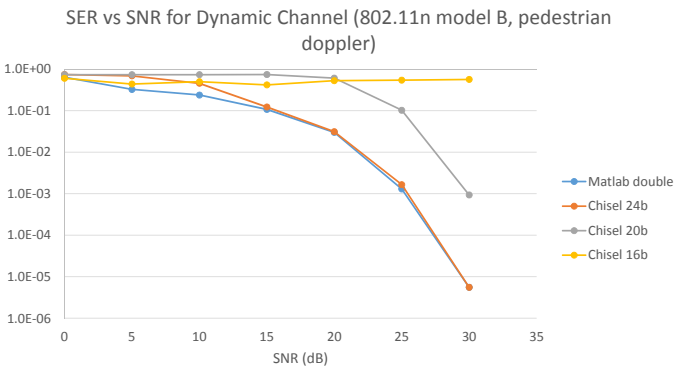

Fig. 8. SER vs SNR curves for Chisel implementations with different bitwidths (dynamic channel)

For the static channel, the 20 bit and 24 bit Chisel implementations very closely track the MATLAB simulations, while the 16 bit simulation is completely incorrect. In the dynamic channel case, higher precision is needed to track the changing channel, and so only the 24 bit implementation is able to match the performance of the MATLAB simulation. Therefore, 24 bits was selected as the desired bitwidth for the design.

The poor performance at lower bitwidths is caused by arithmetic overflow during the matrix inverse operation in the Initialize Weights module. Better performance can be achieved at lower bitwidths if saturating arithmetic or floating-point were used. Unfortunately, neither of these two features are currently supported in Chisel.

IV. Synthesis Options and Constraints

The clock rate at which the design was synthesized was determined by maximum throughput requirements for 802.11ac and LTE. The maximum downlink rate for 802.11ac (80MHz channel) is 278 000 symbols/sub-carrier/antenna/second [5]. With 234 sub-carriers, this translates to 70 million symbols/second/antenna.

LTE has a maximum symbol rate of 14 000 symbols/sub-carrier/antenna/second [6] and with 1200 sub-carriers, this results in a throughput of 16.8 million symbols/second/antenna.

If the sub-carriers are decoded sequentially (i.e. time-multiplexing the decoder between the sub-carriers), but all the antenna inputs are decoded in parallel, the Adaptive MIMO Decoder will need to be clocked at 70MHz (i.e. at the 802.11ac symbols/antenna rate). However, it is also possible to process multiple sub-carriers in parallel. The advantage of this approach is that all the sub-carriers can share (on a time-multiplexing basis) common Channel Estimation and Weight Initialization modules, as these modules only run at the beginning of each packet. The only modules that therefore need to be duplicated are the the Matrix Engine and the Adaptive Decoder.

Table I therefore shows the required clock rate for parallelism levels of 1, 2 and 4. Here, the level of parallelism indicates both the number of sub-carriers that can be decoded in parallel, as well as the number of instantiations of the Matrix Engine and Adaptive Decoder. This means that all three scenarios shown in the table have *identical* throughput.

TABLE I
VLSI clock rate for different levels of parallelism

Number of parallel sub-carriers	Clock Rate
1	70 MHz
2	35 MHz
4	17.5 MHz

Estimating the channel matrix at the beginning of each packet and computing the resulting weight matrix takes only 50 clock cycles. Since it takes at least 45000 cycles to decode an entire packet, the impact of these intialization operations on latency and throughput was easily minimized with a small amount of buffering and a very slight increase in clock speed.

V. Synthesis Results

The Chisel designed was synthesized at the three clock rates shown in Table I. For the 35 MHz and 17.5 MHz cases, the Matrix Engine and Adaptive Decoder modules were replicated the required number of times. The results, for a 4 antenna decoder, are shown in Figure 9. Again, it must be emphasized that all three synthesized versions have the same throughput, and are able to meet the requirements for 802.11ac and LTE.

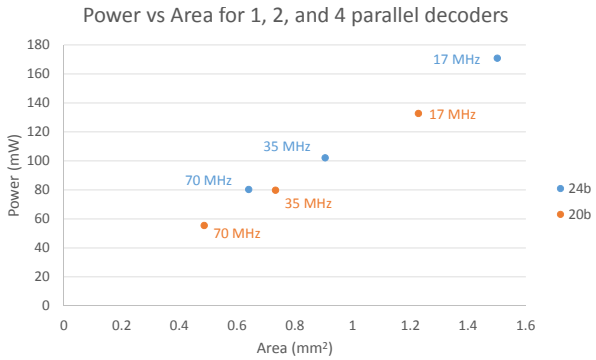

Fig. 9. Power vs area for different levels of parallelism

It is quite clear that the 24-bit 70MHz version results in the smallest area (0.6mm^2) and lowest power (78mW). While the 20-bit version does result in a slightly smaller design, it cannot be used due to poor reliability in changing channel conditions. While lower power figures at the slower clock rates could be achieved if voltage scaling was used, this feature was not available to us.

Extrapolating from Figure 9 shows that even smaller area/power figures could be obtained if the data streams from the 4 antennas were processed sequentially (rather than in parallel, as they are currently done), and the clock rate was increased accordingly.

The power consumed by each module in the MIMO decoder is shown in Table II and illustrated in Figure 10. These show that almost half the power is consumed by the initialization modules: Channel Estimator and Initialize Weights. Since these modules are used for $< 0.1\%$ of the total operation time, power gating them will reduce the average power consumption to 42mW.

ANTONIO: if you can, please insert some 2x2 vs 4x4 results here. Mention that 4x4 hardware supports 2x2, but more efficient to generate 2x2 hardware.

TABLE II
BREAKDOWN OF POWER BY MODULE

Module	Submodule		Power (mW)
Channel Estimator			2.13
Initialize Weights	Mat4 Inverse	3.96	
	Mat2 Inverse	24.8	
	Module total		33.7
Matrix Engine			25.4
Adaptive Decoder			15.1
Queues			1.81
Total			*78.14*
Total without initialization modules			42.31

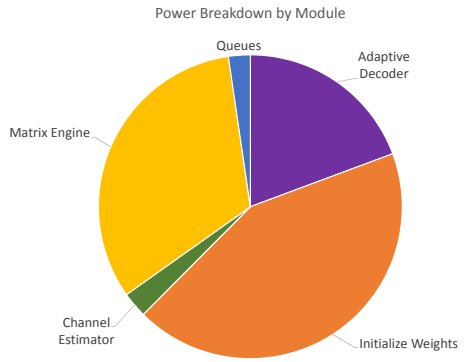

Fig. 10. Power of LMS MIMO Decoder by module

VI. CONCLUSIONS AND FUTURE WORK

In this project, we have demonstrated an adaptive LMS-MMSE MIMO decoder that can achieve the throughput requirements of LTE and 802.11ac using QPSK modulation. The final design takes up 0.6mm^2 of area and consumes 42mW when including power gating of the initialization blocks. By exploiting the LMS adaptive loop, this design is also able to track slow channel variations while maintaining good performance.

There are several areas of further work that could improve this design:

1) Implement power gating to turn off the initialization modules when they are not in use.
2) Reduce the bit widths in the main adaptive decoder, while keeping the 24-bit words in the initialization blocks (where the matrix inversion is sensitive to precision errors), in order to decrease area and power. Furthermore, the required bit width in the inversion block could also be reduced by using floating point or saturating arithmetic.
3) Evaluate energy reduction techniques when using parallel decoding modules. Since each module operates at a lower clock rate, it should be possible to reduce energy per decoded symbol by reducing the supply voltage.
4) Design a control loop to dynamically select the LMS step size based on the time scale of channel variations.
5) Integrate this MIMO receiver with a channel decoder and use the decoder hard decisions to adapt the receive filter.

REFERENCES

[1] D. Tse, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.
[2] J. Litva and T. Lo, *Digital Beamforming in Wireless Communications*. Artech House Publishers, 1996.
[3] R. Adve, "Optimal Beamforming. ECE1515S: Smart Antennas Course Notes," May 2007, [Online]. Available: http://www.comm.utoronto.ca/ rsadve/Notes/BeamForming.pdf.
[4] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 1216–1225.
[5] M. Gast, *802.11ac: A Survival Guide*. O'Reilly Media, 2013.
[6] I. Poole, "LTE MIMO: Multiple Input Multiple Output Tutorial," May 2014, [Online]. Available: http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/lte-mimo.php.