My Implementation of 802.11-2012

*and why I now regret many life decisions

Jack Spencer "Danger" Langner
Dept. of Electrical Engineering
The Cooper Union
Manhattan, New York
langner@cooper.edu

***Abstract*—This document details the details of IEEE 802.11-2012 that I implemented as part of the first real project in *ECE408: Wireless Communications*. I also include some of the difficulties encountered and certain parts that were not implemented. Finally, the bit error rate (BER) curves different modes of sending data in a noisy channel are presented following a simulation. Additionally, I simulated sending a text file through the AWGN channel, those results are not shown, but it was successful.**

***Index Terms*—map**

I. Introduction

The first project for ECE408: Wireless Communications was to implement a portion of a wireless standard. I chose to focus on the IEEE standard 802.11 which deals with the media access control (MAC) and physical layer (PHY) protocols that are used in wireless local area networks (WLAN). Rolled out in 1997, 802.11 has had amendments to improve the quality of service for devices on a network as both better techniques and technologies become available. This has come to include the use of orthogonal frequency-division multiplexing (OFDM) waveforms, low density parity check (LDPC) error correcting codes, and even beamforming. While these are interesting features to implement in a complete system that has to handle multiple users, some of these utilities are beyond the current scope of the course. To this end, my goal was to implement the short (and long) preamble, the signal frame, and data frame(s) complete with MAC header and cyclic redundancy check. All this with the appropriate error correction, interleaving, scrambling and modulation including OFDM. The standard that this is based off of is 802.11-2012, this is a roll-up standard, so nothing was added, but the previous standards in the 802.11 family were condensed into a single document to ensure compatibility and for ease of use.

II. 802.11 Preambles

In 802.11-2012 there are two preambles, short and long. The two preambles are the first component of a message that is sent and then received. They are used to establish signal detection, automatic gain control, diversity selection, timing synchronization, and channel and frequency offset estimation. The short preamble makes use of 12 of the subcarriers, out of the total of 52, with two symbols $1 + j$ and $-1 - j$, where j is the imaginary unit. The short preamble is repeated 10 times with 160 samples in time. This done because if a 20 MHz cubcarrier spacing is assumed, then the 160 time samples divided by the subcarrier spacing yields a message with a length of 8 μs in time, which helps reduce the intersymbol intereference (ISI) of the channel. The long preamble has symbols on all 52 subcarriers where each subcarrier contains a binary phase shift keying symbol, i.e. $+1$ or -1. These 52 subcarriers plus a DC null are used to generate 64 samples in time via an inverse Fast Fourier transform (IFFT). In order to occupy the same amount of time as the 10 short preambles, the long preamble is sent twice, but this only accounts for $2 \times 64 = 128$ samples in time. In order to compensate for this, 2 guard intervals are placed in front of the long preamble time samples to make up the remaining 32 samples, each guard interval is 16 time samples. So the timing of the preambles, is 0.8 μs per short preamble, 1.6 μs per guard interval, and 3.2 μs per long preamble. Since the preambles are used for timing purposes and given that I could not figure out how the standard implemented the IFFT, I did not include the preambles in the actual data transmission because luckily I was not building the hardware responsible for transmitting and receiving data. If I was though, the preambles would be my best friends and I would thank the IEEE for including them in the standard.

Identify applicable funding agency here. If none, delete this.

III. The Signal Frame

After the proper synchronization has been performed with the preambles, the signal frame is processed. The signal frame carries meta data about the information that follows it, specifically the signal frame contains the modulation and coding scheme (MCS), a reserved bit, the length of the message to be received in number of bytes/octets, a parity bit and 6 bit tail.

Rate (4 bits)				Reserved	LENGTH (12 bits)											Parity	SIGNAL TAIL					
R1	R2	R3	R4	R	LSB										MSB	P	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Fig. 1. Composition of SIGNAL frame

A. Encoding

From the above, it is seen that the signal frame contains 24 information bits. The index of each bit is given along with its

meaning and the field to which it belongs. Note, the Parity bit is calculated by taking the sum of the Rate and Length bits modulo 2. Then the frame is encoded with a rate 1/2 convolutional encoder. The polynomials that are used for the encoding are $g_0 = 133_8$ and $g_1 = 171_8$, so both polynomials have a constraint length of 7.

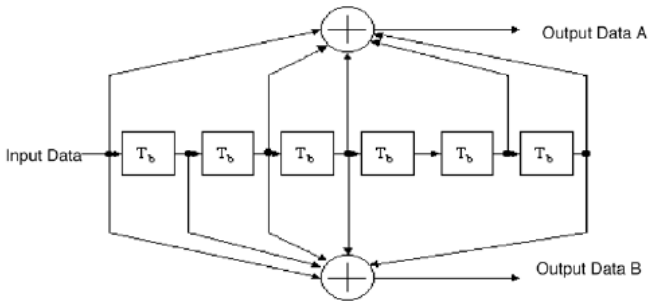

Fig. 2. Visualization of convolutional Encoded connections

The output of the encoder works as follows, 1 bit is fed in and based on the state of the register, 2 outputs are produced in parallel. An example, suppose a 1 is being fed in with the register in the all zero state, then output of both sums (modulo 2) would be 1, then another 1 is fed in resulting in a 1 from A and a 0 from B, and then a third 1 is fed in leading to a zero from A and a 1 from B. So the sequence $[1_1 1_2 1_3]$ gets mapped to $[1_{A,1} 1_{B,1} 1_{A,2} 0_{B,2} 0_{A,3} 1_{B,3}]$

B. Interleaving

After encoding the signal frame, the coded bits are interleaved twice. The first round of interleaving is done so that adjacent bits are not mapped onto the same constellation symbol and therefore different subcarriers. After this, the bits are interleaved again so that adjacent bits are mapped differently onto the constellation to avoid long runs of low reliability least significant bits (LSB's) being received. The interleaving is done according to the following functions, where k is the initial index of the bit, i is the index after the first interleave, and j is the index after the second interleave.

$$
\begin{aligned}
i &= (N_{CBPS}/16)(k \mod 16) + \lfloor k/16 \rfloor \\
j &= s \times \lfloor i/s \rfloor + \left(i + N_{CBPS} - \left\lfloor \frac{16i}{N_{CBPS}} \right\rfloor \right) \\
s &= \max(N_{BPSC}/2, 1)
\end{aligned}
$$

where k, i, $j \in \{0, N_{CBPS} - 1\}$. Additionally, s is gives information on how many bits will make up the real and imaginary part of the modulated symbol,N_{BPSC} is the number of bits per subcarrier (or equivalently the number of bits per modulated symbol), N_{CBPS} is the number of coded bits per OFDM symbol, and $\lfloor \cdot \rfloor$ represents the floor function. In order to recover the correct order of the bits on the receive side, the inverse mapping is given as:

$$
\begin{aligned}
i &= s \times \lfloor j/s \rfloor + \left(j + \left\lfloor \frac{16 \times j}{N_{CBPS}} \right\rfloor \right) \mod s \\
k &= (16 \times i) - (N_{CBPS} - 1) \left\lfloor \frac{16 \times i}{N_{CBPS}} \right\rfloor
\end{aligned}
$$

with s defined as before. Note the dependence the interleaver has on the number of coded bits per OFDM symbol, this means as the number of bits changes, so too does the indexing of the interleaver.

C. Modulation

Following the encoding and interleaving, the bits are then modulated according to a binary phase shift keying (BPSK) scheme. In 802.11-2012, BPSK is implemented where a "0" bit is mapped to -1 and a "1" bit is mapped to 1. The -1 and 1 symbols are what I have been referring to as modulated symbols. Under the construction of the SIGNAL frame, there are 48 modulated symbols which get mapped to the 48 subcarriers of the OFDM waveform. In 802.11-2012, the OFDM waveform is formed by taking a 64 point Inverse Fast Fourier Transform (IFFT) of the 48 modulated symbols plus 4 pilot tones and null at the 0^{th} index. This raises the question of how the symbols are mapped to the IFFT, and it is immediately answered by the definition of $M(k)$ which is defined as:

$$
M(k) = \begin{cases}
k - 26 & 0 \leq k \leq 4 \\
k - 25 & 5 \leq k \leq 17 \\
k - 24 & 18 \leq k \leq 23 \\
k - 23 & 24 \leq k \leq 29 \\
k - 22 & 30 \leq k \leq 42 \\
k - 21 & 43 \leq k \leq 47
\end{cases}
$$

which maps $\{0, 47\} \longrightarrow \{-26, 26\}$, where the 0^{th} index is filled in with 0. Positions -21, -7, 7, and 21 are occupied by the pilot tones which have values of 1, 1, 1, and -1 with respect to the ordering of the indices. Furthermore, the polarity of the pilot tones are determined by setting the scrambler to the all 1's state and then mapping 1 to -1 and 0 to 1, and using this to scale all four of the tones. Additionally, each output of the scrambler only scales the pilot tones of one OFDM waveform, so as an example assume two consecutive outputs of the scrambler are 0 and 1, so the scale factors are 1 and -1, respectively. For the first OFDM waveform being produced the pilot tones will be [1 1 1 -1] and in the second the tones will appear as [-1 -1 -1 1]. The jump from 53 symbols (in frequency) to 64 samples (in time) adds a guard band to help combat intersymbol interference (ISI). After the 64 point IFFT is performed, the first 16 samples are appended to the OFDM waveform, so that the aggregate waveform has 80 samples, and assuming that there is 20 MHz spacing in the channel, the waveform containing the data will last for 4 μs. The standard did a poor job of defining exactly how to implement the IFFT, so I just used the built in OFDM modulator in the communications toolbox, sad.

D. The Reverse Process

After the data was OFDM modulated, I put it through an additive, white, Gaussian noise (AWGN) channel. The received data was then put through the built in OFDM demodulator which removes the cyclic prefix and performs the FFT. Then the DC null and pilot tones were removed and the data was BPSK demodulated. Then the binary data was deinterleaved using the formulas given above and then it was decoded using the Viterbi algorithm with a hard decision rule and a traceback depth of 18.

IV. DATA FRAMES

A good portion of the implementation of 802.11-2012 is handled within the SIGNAL frame, but there is still more to do with the DATA frames that follow the SIGNAL frame. Remember how the first four bits of the SIGNAL frame determine the data rate, well in 802.11-2012, when you're using one spatial stream there are 8 options, which are listed here now

TABLE I
DATA RATE OPTIONS

Rate	Bit	Modulation	Code Rate	N_{BPSC}	N_{CBPS}
6	1101	BPSK	1/2	1	48
9	1111	BPSK	3/4	1	48
12	0101	QPSK	1/2	2	96
18	0111	QPSK	3/4	2	96
24	1001	16-QAM	1/2	4	192
36	1011	16-QAM	3/4	4	192
48	0001	64-QAM	2/3	6	288
54	0011	64-QAM	3/4	6	288

The table now requires some explaining. First, the Rate column columns refers to the data rate with which the data is being transmitted, it has units of megabites per second [Mbps]. The Bit column is the bit pattern that would appear in the SIGNAL frame. Next, N_{BPSC} stands for number of coded bits per OFDM subcarrier, N_{CBPS} is the number of coded bits per OFDM symbol (which is 48 times N_{BPSC}). Using the code rate and N_{CBPS} the number of data bits per symbol (N_{DBPS}) can be calculated using the following

$$N_{DBPS} = CodeRate \times N_{CBPS}$$

Something else that is useful to know a beforehand is the number of OFDM symbols which will make up the message, so given the number of bits in the DATA frame, the total number of OFDM symbols is given by

$$numSym = \left\lceil \frac{16 + 8 \times numBytes + 6}{N_{DBPS}} \right\rceil$$

where $numSym$ is the number of OFDM symbols/waveforms, $numBytes$ is the number of bytes/octets in the DATA frame, and $\lceil \cdot \rceil$ represents the ceiling function. There is a 16 in the above equation which accounts for the 16 bits that make up the SERVICE field. The first seven bits of the SERVICE field are supposed to contain the initial state of the scrambler that was used, but in reality these seven bits are zeros and it is up to the receiver to figure out the initial state of the scrambler. The other nine bites are reserved and set to 0. The $+6$ is there because the DATA frame is required to have at least 6 tail bits that are all zero. The division by N_{DBPS} and the use of the ceiling function unsure that a integer number of OFDM symbols are generated by the data.

Also, the DATA frame contains the media access control (MAC) header which occupies anywhere between 10 and 36 bytes at the top of the DATA frame, but after the SERVICE field. Additionally, there is a cyclic redundancy check (CRC) that is based on the following polynomial,

$$\begin{aligned} G(x) =& x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} \\ &+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \end{aligned}$$

which is determines the last 4 meaningful bytes of the DATA frame.

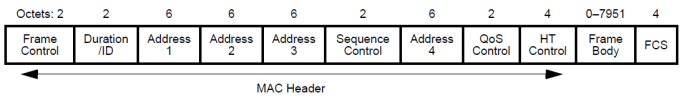

Fig. 3. Components of the MAC header

Briefly, the MAC header is required to contain the Frame Control, Duration/ID, and Address 1 fields as well as the CRC. Here Frame Body is the actual data to be sent/ received. The components of the MAC header have to do with the real world, so I did not implement it in a meaningful way, but given the information, it could be incorporated easily with everything else. Further details on the meaning of the different sections of the MAC header and how to fill them out are given in section 8 of 802.11-2012.

A. Scrambling

In the DATA frame, the first operation performed on the data is scrambling the data. This is accomplished with the use of a linear feedback shift register (LFSR) that has length 7. Within the 802.11 standard, the scrambler is defined by the polynomial $x^7 + x^4 + 1$, which means that the the first, fourth, and seventh bits of the LFSR are added together (modulo 2) and the result is the new bit in the Data sequence. However, I implemented the scrambler by adding the fourth and seventh bits together, shifting the result into the first position of the LFSR while storing the result of the addition to a sequence that has the same length as the DATA bits. Then the DATA bits were added together with the sequence generated by the LFSR. Furthermore, since the sequence was stored in memory for the duration of a single run, I could use the sequence at the end of the receiver side to recover the original message because I was not sure how to figure out the descrambling without knowledge of the initial state while not doing a grid search of all 128 possible initial states.

B. Code Rate

The observant reader would have noticed the in table I different code rates, but I did mention the polynomials that

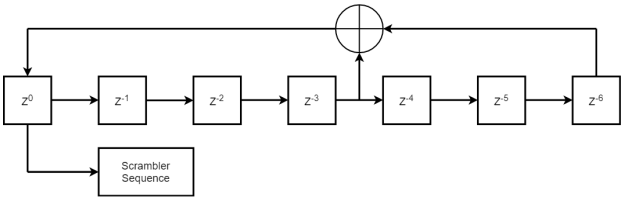

Fig. 4. Implementation of the scrambler

were used to implement these rates. Fear not, as I explain with my soothing, baritone voice. The rate 1/2 code that was defined before in section III-A is used for all the coding rates but a puncturing pattern is employed in order to achieve higher rates. Puncturing patterns can be thought of as a mask which is placed over the encoded bit sequence, I will use a 1 to denote bits that are retained by the puncturing pattern and 0 for bits that are dropped. The mask that achieves a rate of 2/3 is $[1110]$. Using the encoding example above with the first 2 bits ($[1_1 1_2]$) that are fed in, we would get an output of $[1_{A,1} 1_{B1} 1_{A,2}]$ because the second output from B is dropped by the puncturing pattern. Furthermore, it is clear that two input bits lead three output bits with the use of the mask justifying the change of coding rate. Similarly, if all bits from the example ($[1_1 1_2 1_3]$) are considered with the mask $[111001]$, the output is $[1_{A,1} 1_{B1} 1_{A,2} 1_{B,3}]$. Decoding was handled by the built in MATLAB object comm.ViterbiDecoder in all cases and setting the appropriate properties based on the code rate selected.

C. Data Formatting

In the following section, I assume that the binary data has already been scrambled and encoded, the process described here is how I handled the data prior to interleaving. This is necessary because the functions that handle the indices for interleaving are only valid for $i, k, j \in \{0, N_{CBPS} - 1\}$. Also, data vector refers to the entire data frame laid out in a column vector.

As shown in Table I, each data rate has a prescribed N_{BPSC} and N_{CBPS}, which will be used to transform the data vector into a matrix of modulated symbols. First, the data vector is reshaped into a matrix with dimensions $N_{CBPS} \times numSym$, where the first column contains the data with indices from 0 to $N_{CBPS} - 1$, the second column contains the indices from N_{CBPS} to $2 \times N_{CBPS} - 1$, and so on. Now consider only the first column of the new matrix, this column vector is then reshaped into a matrix with dimensions $N_{BPSC} \times 48$. The first column contains the data indices from 0 to $N_{BPSC} - 1$, the second column contains the indices N_{BPSC} to $2 \times N_{BPSC} - 1$, and so on. Then the transpose of this matrix was used, as the MATLAB function $de2bi$ operates on the rows of a binary matrix. This process was repeated for each column of the initial reshaping matrix, resulting in a 3D array with dimensions $48 \times N_{BPSC} \times numSym$. Treating, each $N_{BPSC} \times 48$ slice independently, $bi2de$ was used to convert each row to a decimal number, resulting in a 48×1 vector which represents a single OFDM waveform. Note,by default $bi2de$ treats the first element of the row as the least significant bit, this can be changed but you have to stay consistent between the transmit and receive sides. Following the conversion to decimal, the resulting column vectors were stored in matrix with dimension $48 \times numSym$ and then each element was modulated according to the scheme prescribed by the data rate.

As an aside, 802.11-2012 describes how the data blocks of size N_{BPSC} should be coded relative to the scheme prescribed, but this felt like a minor detail that would not add to my understand of the standard if implemented. Therefore, I used default constellations which may effect the bit error rate of a given scheme, but I really don't believe this to be true.

V. Results

As per the syllabus of the class, the deliverable of the project should consist of a working link in AWGN and a BER curve, and this is exactly what I deliver on the next page in Fig. 5. I was able to simulate all 8 data rates along with a theoretical BER curves for uncoded BPSK and BPSK using the rate 1/2 convolution encoder that was defined in section III-A. Here's my interpretation of the results. Sending data at higher rates comes with the penalty of higher BER, so for these data rates to be useful the environment has to be nice. This stems from the fact that the constellations implemented in 802.11-2012 all have unit average energy, so BPSK has a distance of 2 between symbols, whereas 64-QAM has a minimum distance of 0.3084, inherently making errors more likely. This is better visualized with the constellations overlaid each other.

Formatting pictures in LaTeXis not fun man. Back to the relevant discussion. Additionally, it can be seen that that data rates of 6 Mbps and 12 Mbps perform essentially identically and the same can said for 9 Mbps and 18 Mbps. It should be noted that this applies to a channel where only AWGN is applied, if the noise had a different distribution or the channel introduced ISI, this result would probably no longer be true. Another interesting detail to point out, is that both 6 Mbps and 12 Mbps perform better than the theoretical encoded BPSK. This boost in performance is mostly like to the interleaving which reduces the number of burst errors. With fewer burst errors, the Viterbi algorithm will have increased success. In real world terms this manifests itself as a 1 dB coding gain at a BER of 10^{-3} for including interleaving.

The results presented here are based off of only 50 iterations at each E_B/N_0 level. Ideally more iterations would be run, but this simulation took nearly 2.5 hours and I was using what I considered to be a reasonable block size 0f 10 kilobytes, because it felt right. I implemented another version that supports text file input and that does the proper encoding and such to show what could happen data is transmitted.

Acknowledgements

As per usual, I would like to thank Karol Wadolowski for his continued friendship and specifically for helping me figure out how the scrambler works. Also, the kind strangers on

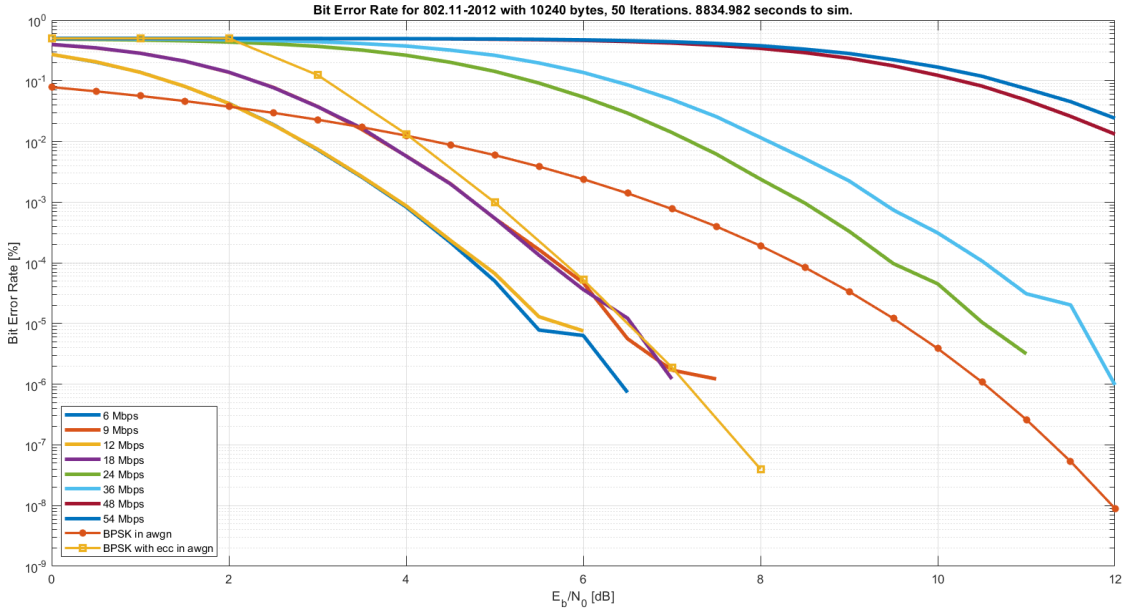

Fig. 5. BER curves for MCS's of 802.11-2012

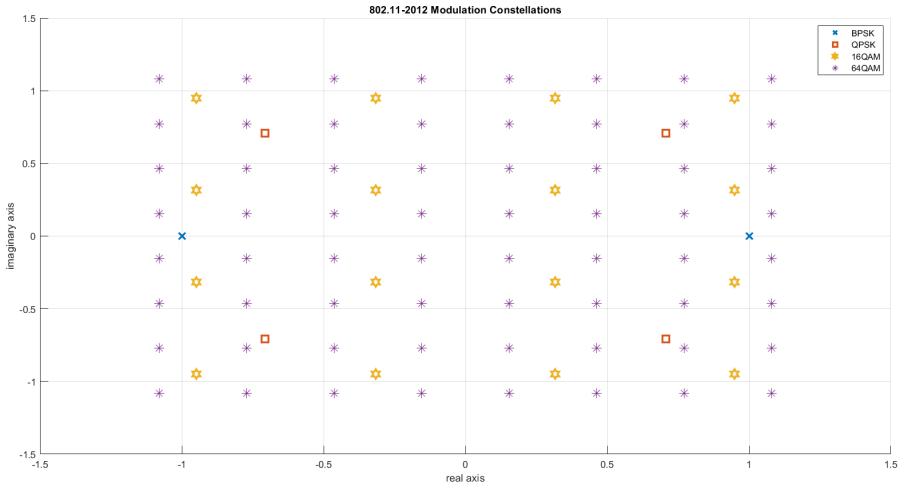

Fig. 6. Constellations of 802.11-2012

the internet that answered the questions I had about overleaf and LaTeXeven before I knew I had these questions. Finally, I would like to thank Hilary Hahn, who is a supremely talented concert violinist whose recorded performances helped me to focus throughout the duration of this project. If you have the time checkout this rendition of Mendelssohn's violin concerto.

References

[1] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007) , vol., no., pp.1-2793, 29 March 2012