A Parametrically Generated OFDM Baseband Modem

Sean Huang, *Student Member, IEEE*, Kunmo Kim, *Student Member, IEEE*, Paul Kwon, *Student Member, IEEE*, Joshua Sanz, *Student Member, IEEE*, Meng Wei, *Student Member, IEEE*[1]

***Abstract*—Orthogonal Frequency Division Multiplexing (OFDM) radio systems such as Wi-Fi and LTE require significant digital signal processing (DSP) at baseband before transmission and after reception, and in order to meet latency and power requirements they must be implemented in dedicated hardware [1]. State-of-the-art baseband modems require many hundreds of thousands of lines of Verilog in order to support even one standard. [2]. In this work, we present a baseband modem written in Chisel [3] which provides the ability to generate baseband configurations for multiple standards from one concise set of code. The functional blocks are verified at the unit level with multiple parameter sets, demonstrating the flexibility of the generators. A modem which supports IEEE 802.11a baseband processing was generated using far fewer lines of code than a typical Verilog implementation, and with the ability to reconfigure for other OFDM radio standards.**

***Index Terms*—orthogonal frequency division multiplexing, OFDM, baseband processing, radio, WiFi, Viterbi algorithm, zero-forcing, raised-cosine, QAM, convolutional coding, carrier frequency offset.**

I. Introduction

ORTHOGONAL Frequency Division Multiplexing (OFDM) has become ubiquitous in almost all everyday connected devices in the form of Wi-Fi and LTE. In each of these devices, considerable processing power is devoted to performing back-end calculations and signal processing for the OFDM systems. At the ever-increasing speeds of modern communication, it is necessary to have hardware dedicated solely to performing the digital signal processing for transmission and reception of OFDM [1].

However, the complexity of the processing blocks required in the entire OFDM transceiver system necessitates a very involved design process, and can require many thousands of lines of Verilog [2]. Not only does this present a significant obstacle to the initial development of the OFDM back-end, but designing around another standard usually requires completely redesigning blocks and rewriting the thousands of lines of hardware description language to implement the new standard, despite the blocks still performing the mostly the same calculations. It would be a significant improvement to productivity if the back-end design could be made more general and scripted such that any design for any standard could be generated simply by adjusting some input parameters.

[1]S. Huang, K. Kim, P. Kwon, J. Sanz, and M. Wei are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA. e-mail: ([sehuang, kunmok, hunjaekwon, jsanz, meng_wei]@berkeley.edu).

Manuscript submitted Dec. 5, 2018.

II. Current State of the Art

The OpenOFDM project [2] demonstrates the level of design effort needed to create a working OFDM back-end. The project contains more than 800,000 lines of Verilog either written by the main contributor or as existing Xilinx IP. Since the difference in block operation for the different standards is minimal, the majority of the work was to just bring up a working back-end for even one of those standards. Having a generator to create the baseband modem requires much less code verbosity and can streamline the redundant Verilog code that is currently used to design these blocks.

A generator would also allow for easy reuse and modification of blocks, as existing Verilog is hard to modify for evolving specifications. A new designer inheriting an existing Verilog design must decipher the thousands of lines of code, then determine how to modify the design to accept the new standard, or rewrite the entire block from the beginning. With a generator, large blocks of redundant code can be abbreviated into a more readable loop or function, and because the generator preserves the procedure of writing the module, modifying it to change functionality is simple and the new functions integrate seamlessly with the original design.

III. Implementation

This work uses the Chisel hardware construction language to write generators for an OFDM transceiver supporting IEEE 802.11a standards. The transceiver was written as two independent processing chains, one for transmitting and one for receiving. The two chains are shown in Figures 1 and 4. Detailed descriptions of the blocks and the implemented algorithms are provided in Sections IV and V.

IV. Transmitter

A. Raised Cosine Filter

The transmitter implements a parameterized raised cosine pulse-shaping filter [4]. The bandwidth extension parameter α, number of pulses spanned $npulse$, and number of samples per symbol $nsps$, are all parameterizable. The tap generating function is given below.

$$p[n] = \frac{\text{sinc}\left(\frac{n}{nsps}\right) * \cos\left(\frac{\pi\alpha n}{nsps}\right)}{1 - \left(\frac{2\alpha n}{nsps}\right)^2}$$

$$n = -(nsps \times npulse), \ldots, (nsps \times npulse)$$

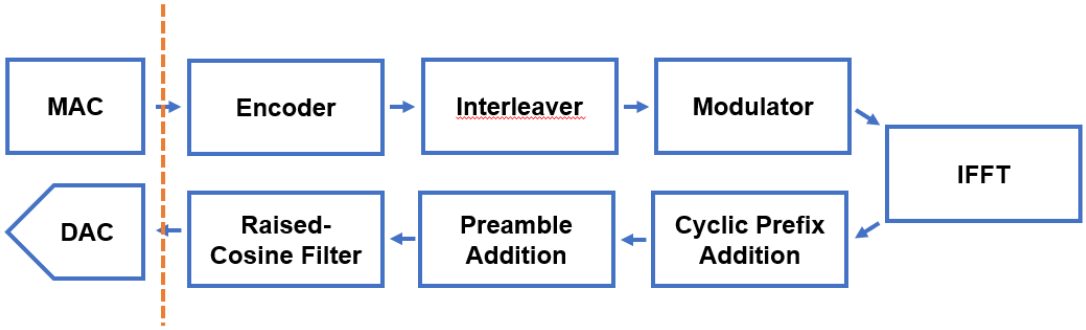

Fig. 1. 802.11a transmitter PHY block diagram

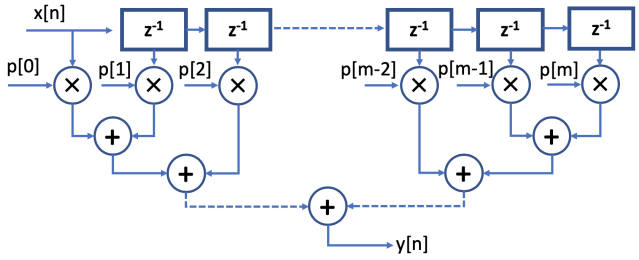

Fig. 2. Direct form FIR with adder tree

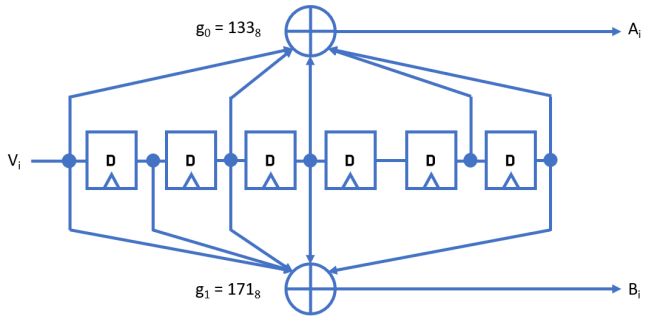

Fig. 3. 802.11a convolutional encoder

The filter is applied by a direct-form finite impulse response (FIR) filter whose taps are generated by the raised cosine parameters. The architecture is shown in Figure 2.

B. Transmit Encoder

1) Convolutional Encoding: Convolutional encoding is a type of error correction coding that generates parity symbols by creating time-invariant trellis. Each clock cycle the encoder takes a bit from serializer and store it in a shift register. Values stored in shift registers are fed to multiple-input XOR gate and generates codewords. The generator polynomial determines which flip flop outputs to be connected to the XOR gate, and must be chosen carefully to avoid catastrophic encoding while maximizing free distance. For instance, generator polynomial of (7,5) provides slightly larger free distance compared to generator polynomial of (7,6) at the same hardware complexity level. Convolutional encoder for 802.11a standard is shown in Fig. 3

Since the decoding process requires distinct minimum path metric at the end of trellis, normally convolutional coding requires padding 'm' zeros at the end of message. This can potentially results in a significant coding rate loss for short input sequence length. This coding rate loss can be generalized as

$$\mathrm{R_{eff}} = \mathrm{R_{ideal}} \cdot \frac{\mathrm{L}}{\mathrm{L}+m}$$

where L is the length of the input sequence and m is the number of memory elements.

To avoid such coding rate loss, tail-biting has been adopted [5]. Tail-biting, however, is not a requirement for 802.11a.

TABLE I
DATA RATE V.S. CODING RATE AND MODULATION SCHEME

Rate field	Data Rate (Mbps)	Coding Rate	Modulation Scheme
1,1,0,1	6	1/2	BPSK
1,1,1,1	9	3/4	BPSK
0,1,0,1	12	1/2	QPSK
0,1,1,1	18	3/4	QPSK
1,0,0,1	24	1/2	QAM-16
1,0,1,1	36	3/4	QAM-16
0,0,0,1	48	2/3	QAM-64
0,0,1,1	54	3/4	QAM-64

Tail-biting encoding ensures that the starting state of the encoder is the same as its ending state, and therefore it allows circular convolutional decoding. In other words, decoding can be initiated regardless of its starting position. In addition, $\mathrm{L}/(\mathrm{L}+m)$ coding rate loss can be eliminated. However, if the transmitter is guaranteed to send out long enough sequence, tail-biting is not a necessary features to be implemented.

2) Puncturing: Puncturing block enhances the coding rate by ignoring a group of selected codes at the cost of coding gain. Puncturing matrix must be chosen carefully to avoid significant BER loss at a given SNR. Table I shows the relationship between puncturing matrix and coding rate and modulation scheme. Puncturing block must have a capability of changing its coding rate in real time once the header transmission is over.

Encoder block takes input serialized bitstream from MAC layer. MAC layer transmits two signal groups: Data and Control. Data port sends serialized bits to be encoded. Control signal sends multiple different signals. `pktStart` and `pktEnd` indicate the start and end of 802.11a PPDU frame

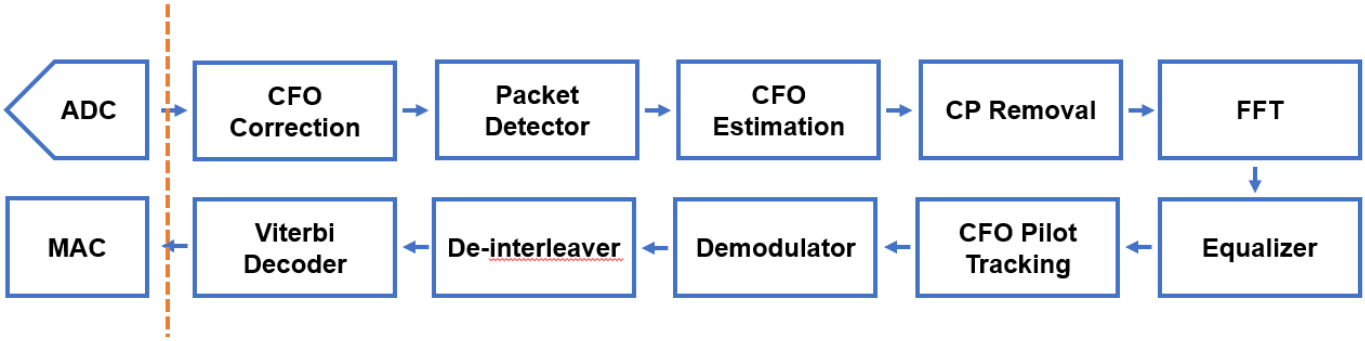

Fig. 4. 802.11a receiver PHY block diagram

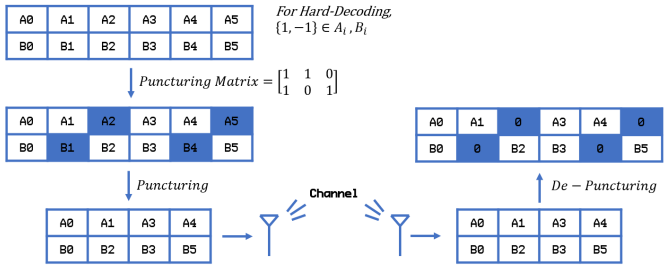

Fig. 5. Puncturing and De-Puncturing process

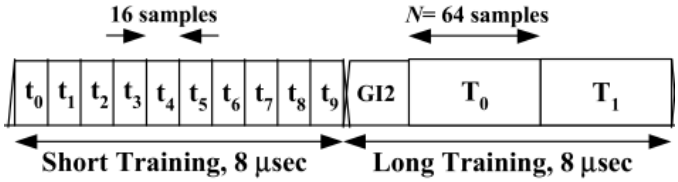

Fig. 6. 802.11a OFDM Training Fields [6]

as a pulse. Encoder must ensure that `pktStart` and `pktEnd` are synchronized with bits from data port at the output.

MAC control also sends `isHead` and puncturing matrix (Table I). `isHead` signal indicates whether the currently receiving bits from data port is header information or part of PSDU. When the header information is coming through the FIFO, encoder fixes its coding rate to $1/2$ and force the modulation scheme to be BPSK. Once it starts receiving PSDU, puncturing module changes the coding rate to the proper value specified in the header information, and send corresponding modulator scheme to the signal mapper immediately.

V. Receiver

A. Carrier Frequency Offset

In any heterodyne receiver, the local frequency of the receiver is never perfectly in phase with the carrier, and so there will be a phase error that accumulates with each sample received. To correct for this, a carrier frequency offset estimation and correction block is included in the receive chain of the OFDM transceiver.

1) Estimation: In the 802.11a standard, each OFDM frame begins with a set of training fields. These are a repeating sequence of samples. In the coarse CFO estimation, the short training field is used. This field contains 10 repeating sets of 16 samples each. In this work, the coarse estimation is performed by the method described in [6]

$$\hat{\alpha}_{ST} = \frac{1}{16}\angle\left(\sum_{m=0}^{159} S_m^* S_{m+16}\right)$$

Where $\hat{\alpha}_{ST}$ is the estimated per sample coarse phase error, and S_i is each input I/Q sample. This accumulates a phase error between each sample and its counterpart 16 samples later, which it finally averages across all 16 samples in each pattern.

Fine offset estimation is done in a very similar procedure. The coarse estimation occurs first, after which the correction factor is determined and given to the offset correction block. The fine estimation takes place after this correction has been done, and is applied to the long training field, which contains 2 repeating sets of 64 samples each, preceded by a guard interval. The estimation then follows the same calculation as in the coarse estimation

$$\hat{\alpha}_{LT} = \frac{1}{64}\angle\left(\sum_{m=0}^{63} S_m^* S_{m+64}\right)$$

2) Correction: In this work, the phase error correction is performed by CORDIC rotation of the input I/Q vector. The base correction factor is given by the estimation block, and the correction module applies an accumulating phase error correction to each subsequent sample it receives.

$$\phi_{i+1} = \begin{cases} (\phi_i + \phi_0) - 2\pi & \phi_i + \phi_0 > \pi \\ (\phi_i + \phi_0) & |\phi_i + \phi_0| \leq \pi \\ (\phi_i + \phi_0) + 2\pi & \phi_i + \phi_0 < -\pi \end{cases}$$

This calculation avoids an infinitely growing phase error as the CORDIC is not able to rotate for angles larger than $|\pi|$. If the calculated phase correction is ever larger than π, it converts the angle into the negative angle equivalent so the CORDIC can still operate.

B. Packet Detector

Incoming packets are detected by the receiver using a simple power threshold with hysteresis. A sliding window of baseband

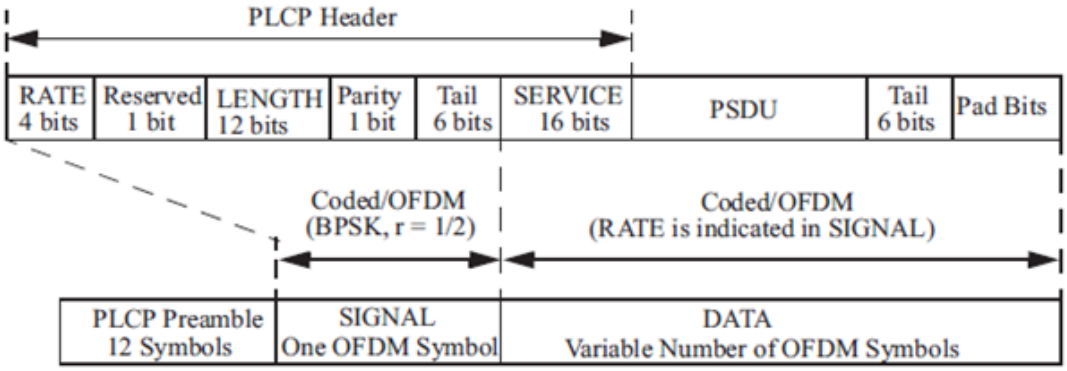

Fig. 7. 802.11a PLCP Frame [7]

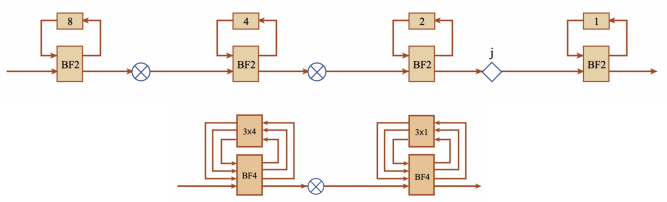

Fig. 8. SDF block diagrams for radix-2 (top) and radix-4 (bottom) [9]

IQ samples is maintained. If all of the samples within the window exceed a parameterized power threshold, a packet is deemed to be detected and the block starts forwarding samples to its neighbor. If, once a packet has been detected, all of the samples within the window fall below the power threshold the packet is deemed to have ended and the block stops sending samples down the chain. The first and last samples have the `pktStart` and `pktEnd` flags asserted, respectively.

C. FFT

The in-place pipeline FFT is implemented as a single-path delay feedback architecture [8]. The generator provides both the radix-2 and radix-4 SDF versions, with radix-4 implementation resulting in less complex multipliers at the expense of more complex adders. The stages of a radix-2 and radix-4 are shown in Fig. 8. This architecture supports both decimation-in-time and decimation-in-frequency decompositions. The FFT also features an unscrambler that reorders the inputs or outputs, based on the user's decomposition setting.

The IFFT reuses the FFT architecture by swapping the real and imaginary components of the inputs and outputs and scaling the outputs by 1/N, where N is the size of the IFFT.

D. Equalizer

The equalizer blocks implements linear zero-forcing equalization [10]. The linear forcing equalizer's transfer function $C(\omega)$ is

$$C(\omega) = \frac{Q(\omega)}{H(\omega)}$$

where $Q(\omega)$ is the transmitted pilot symbol's spectrum and $H(\omega)$ is the received symbol's spectrum. This equalizer completely eliminates inter-symbol interference (ISI) at the cost of greatly amplifying noise in subbands with a small channel response, and thus reducing overall SNR. Figure 9 illustrates this effect.

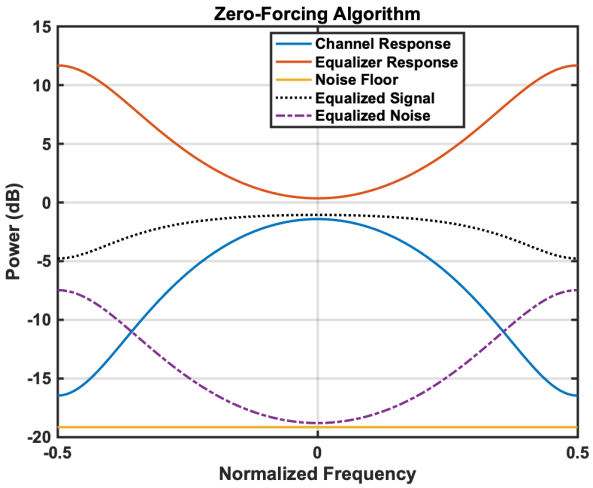

Fig. 9. Zero-forcing equalizer behavior

E. Receive Decoder

The 802.11a receive decoder is composed of following blocks: de-puncturing, header extractor, arbiter, path metric calculator, branch metric calculator, and sliding-window Viterbi decoder. The current Viterbi decoder implemented for this project has capability of computing soft-input branch-metrics. The overall decode architecture is shown in Fig. 10

1) De-Puncturing: De-puncturing module has two distinct purposes. One is buffering the received input sequence to separately decode header and PSDU information. The other is to de-puncture the received bits to decode codewords properly. Punctured bits will be replaced with '0'. This will guarantee that replaced dummy bits have no effect on branch metric calculation. De-puncturing process is described in Fig. 5

2) Header Extractor: Header Extractor module is a simple 48-bit Viterbi decoder that decodes PLCP header information in 62 clock cycles. The frame structure of PLCP header is

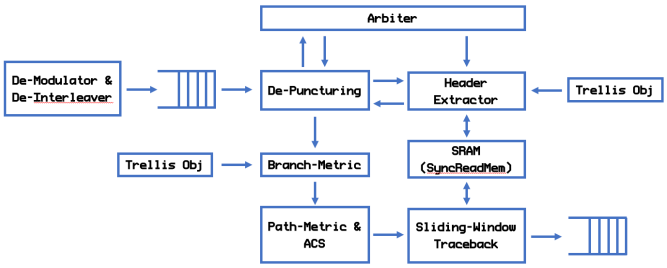

Fig. 10. Rx decoder architecture

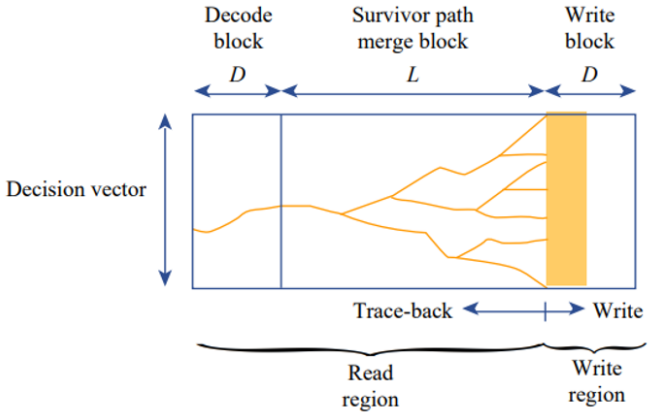

Fig. 11. Concept of sliding window Viterbi decoder [11]

shown in Fig. 7. PLCP header includes a total of 24 bits, but is coded with $1/2$ coding rate and BPSK modulation, yielding a total of 48 bits of encoded PLCP. When the decoder is ready to take the header information, buffered 48 bits of encoded PCLP will be passed to the Head Extractor. Header Extractor stores survival path information into SRAM over 24 clock cycles while accumulating path metrics. After 24 clock cycles, the extractor loads survival paths from SRAM and starts tracing back. Traceback decodes 1 bit at each clock. Head extractor has strict latency requirement that it needs to complete PLCP decoding within 62 clock cycles. This is because de-puncturing and demodulator must be informed of the correct coding rate and modulation scheme before start receiving PSDU frame. As long as the latency requirement is satisfied, then hardware complexity must be minimized at all cost.

3) Arbiter: Arbiter block simply watches over incoming 48 bits from demodulator, and identifies if the received bits are PLCP or PSDU. If it is identified as PLCP, Arbiter block informs De-Puncturing block to pass the buffered 48 bits to Header Extractor. Otherwise, De-Puncturing performs de-puncturing, and then passes n-bit data to path metric over 48 clock cycles.

4) Branch Metric & Path Metric: Branch metric is a measure of the distance between what was transmitted and what was received. Branch metric can take either hard-coded (SInt type) value or soft-coded (FixedPoint type) value, each are called hard-decision and soft-decision, respectively. In the hard-decision case, the branch metric simply finds the minimum hamming distance between received bits and expected bits. In the soft-decision case, the module finds the minimum euclidean distance between received bits and expected bits.

Path metric is a value associated with a state in the trellis. This value is added to the branch metric at each trellis transition, and then accumulates over the trellis. This accumulated path metric is what traceback uses for decoding. Since the traceback module only cares about the minimum accumulated path metric at the end of trellis, the bit width of the path metric can be limited to a reasonably small number. In the current implementation, path metric bit width is set to 5-bit wide. In addition, path metric block checks if the accumulated minimum path metric is higher than 16. If so, then it will subtract 16 from all the elements in path metric registers.

5) Traceback: In the current implementation, data path is decoded via sliding-window Viterbi decoder (SWVD). A simple Viterbi decoder, such as the one used for the header extractor, starts decoding only after it receives the whole bit sequence (48 bit in this case). This simplifies the algorithm but requires larger memory size proportional to the input sequence length. In addition, decoder knows the expected input sequence length only after it receives the PLCP header information. Although the maximum length of PSDU is fixed to 32768 bits, allocating maximum size of memory for any input bit stream is a significant waste of hardware. In order to solve this issue, sliding window Viterbi decoder is deployed. SWVD uses the fact that the survival path merges after 5*K input sequence (Shown in Fig. 11). In Fig. 11, L is defined as 5*K and D is the decoding length. SWVD starts its first decoding L+D clock cycles after it receives the first PSDU bit. After its first encoding is completed, the rest of decoding outputs D bits every D clock cycles. To reduce the latency further, SWVD needs to decode the data while it writes the next incoming bits. This implies that SWVD needs to know how many readports are required to minimize both the hardware complexity and the latency. The minimum number of read port $\mathrm{P_{read}}$ is defined as

$$\mathrm{P_{read}} = \frac{\mathrm{L}-2}{\mathrm{D}} + 2$$

F. Modulator Generator

As shown in Figure 12, the whole modulator consists of BPSK modulator, QPSK modulator and 16QAM modulator. All encoded data bits shall be interleaved by a block interleaver with a block size corresponding to the number of bits in a single OFDM symbol, NCBPS. The interleaver is utilized to avoid long runs of low reliability bits by mapping adjacent coded bits onto nonadjacent subcarriers and mapping adjacent coded bits onto less and more significant bits of the constellation. The Mapper operates at the OFDM symbol level with a block size of (48, 96, or 192) bits. Each block is divided into sub-blocks at the OFDM sub-carrier level. Sub-blocks are of size (1, 2, or 4) bits. The Mapper first converts each sub-block into a complex number representing BPSK, QPSK, or 16-QAM constellation points. The resulting 48 complex pairs are then normalized by KMOD. The output of the modulator and pilots are fed to IFFT block.

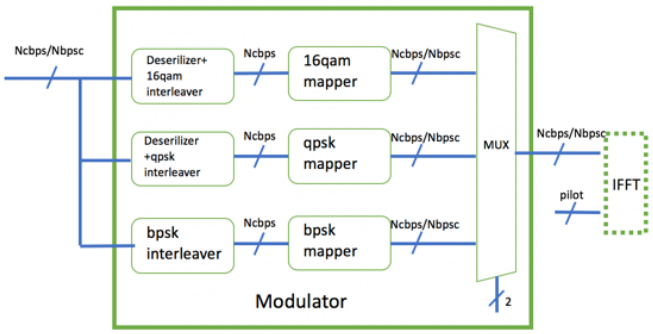

Fig. 12. Block Diagram of Modulator

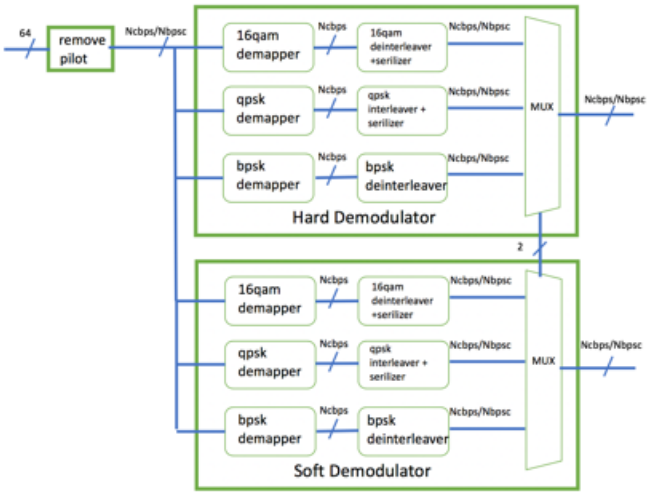

Fig. 13. Block Diagram of Demodulator

G. Demodulator Generator

As shown in Figure 13, both hard demodulator and soft demodulator are integrated in our demodulator. The QPSK and QAM16 demodulator consist of demapper, deinterleaver and serilizer. The BPSK demodulator consists of demapper and deinterleaver. The deinterleaver performs the inverse relation of interleaving operation. In the receiver, bit level demapping can be performed such that the output of demapper is hard, i.e., either a logical value 1 or 0. Alternatively, the demapper output can be soft; a soft-value indicating the probability, that the modulated bit associated with a given demapper output is to be of logical value 1 or 0 .The soft-demapper demaps the received bits into soft bits which have the same sign as provided by a hard detector and whose absolute value indicates the reliability of the decision. Optimal soft-output demapping algorithms involve computationally complex functions such as logarithmic and exponential functions, and thus are not well suited for hardware implementation. In order to reduce the hardware complexity, our soft demapper is based on simplified suboptimal Log-Likelihood Ratio(LLR). The final soft bit values can be calculated as

$$\mathrm{LLR}(b_{I,k}) = \frac{|G_{ch}(i)|^2}{4}\left\{\min|y[i]-\alpha]|^2 - \min|y[i]-\beta]|^2\right\}$$

For $\alpha \in S_0^{I,k}$, and $\beta \in S_1^{I,k}$.

VI. Results and Conclusion

By creating generators in the Chisel hardware construction language, an OFDM transceiver supporting 802.11a Wi-Fi was designed and tested using only 17,415 lines of code. Of those, slightly more than half were the actual implementation at 8,723. Using hardware generators allows for flexibility in supporting standards, such as changing the number of subcarriers and making the design type generic. Compared to the OpenOFDM implementation, which includes many hundreds of thousands of lines of Verilog and does not readily adapt to new standards, our parametrically generated implementation is better suited to the rapidly evolving world of OFDM-based communications.

References

[1] A. Ren, M. Luo, and F. Hu, "FPGA implementation of an OFDM modem," in *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2009)*, pp. 761–764, Dec 2009.

[2] J. Shi, "OpenOFDM." https://github.com/jhshi/openofdm, 2017.

[3] A. Wang, P. Rigge, A. Izraelevitz, C. Markley, J. Bachrach, and B. Nikolić, "ACED: A Hardware Library for Generating DSP Systems," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 61:1–61:6, ACM, 2018.

[4] K. Gentile, "The care and feeding of digital, pulse-shaping filters," *RF DESIGN*, vol. 25, pp. 50–58, 2002.

[5] R. V. Cox and C.-E. W. Sundberg, "An Efficient Adaptive Circular Viterbi Algorithm for Decoding Generalized Tailbiting Convolutional Codes," *IEEE Transactions on Vehicular Technology*, vol. 43, pp. 57–68, Feb. 1994.

[6] E. Sourour, H. El-Ghoroury, and D. McNeill, "Frequency offset estimation and correction in the ieee 802.11a wlan," in *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, vol. 7, pp. 4923–4927 Vol. 7, Sept 2004.

[7] A. Technologies, "802.11a WLAN Signal Studio Software for the ESG-D/DP Series Signal Generators," Tech. Rep. 14730, Agilent Technologies, 10 2002.

[8] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proceedings of International Conference on Parallel Processing*, pp. 766–770, April 1996.

[9] V. Stojanovic, "Lecture 10: Fast Fourier transform: VLSI architectures," in *6.973 Communication System Design*, 2006. MIT OpenCourseWare.

[10] J. W. M. Bergmans, *Linear Equalization*. Springer-Science+Business Media, B.V., 1996.

[11] V. Stojanovic, "Lecture 15: Viterbi algorithm advanced architecture," in *6.973 Communication System Design*, 2006. MIT OpenCourseWare.