FPGA-Based Latency-Insensitive OFDM Pipeline for Wireless Research

James Chacko, Cem Sahin, Danh Nguyen, Doug Pfeil, Nagarajan Kandasamy, and Kapil Dandekar

Drexel Wireless Systems Lab, Electrical and Computer Engineering

Drexel University, Philadelphia, PA 19104

Email:{jjc652, cs486, dnguyen, dsp36, kandasamy, dandekar}@drexel.edu

Abstract—**This paper develops a programmable hardware implementation of the physical layer for cognitive wireless communication systems that use orthogonal frequency division multiplexing (OFDM) schemes. Data-flows within hardware implemented baseband architectures for all communication standards are quite regular in nature, thereby enabling the construction of fast, synchronized and optimized baseband systems on FPGAs. We designed an OFDM pipeline comprising codec, modulation, interleaving, piloting, channel estimation, and IFFT stages in which each stage can be configured at design time or at run time to accommodate different communication standards as well as different configuration settings for a single standard—a key feature necessary for dynamic spectrum sensing and utilization. This flexibility in hardware is achieved by designing each individual stage with room for scaling/modification and having the overall pipeline be insensitive to the latencies incurred by individual pipeline stages. This is done by using stallable stages with centralized control through cross communication between modules both directly and indirectly using code running on the MicroBlaze processor. The OFDM pipeline is implemented on a Xilinx Virtex-6 FPGA and its performance is characterized in terms of functional correctness and FPGA implementation area cost. Experimental results and preliminary simulations of our FPGA based design can run at flexible coding rates of 1/2 and 3/4 with modulation schemes of 4QAM and 16QAM respectively.**

Keywords—*OFDM Pipeline, Experimentation, Algorithms, Physical Layer, Software Defined Radio, Architecture Design*

I. INTRODUCTION

Defining 'software' in software defined radio (SDR) is truly a challenge, and in the extensive research done in this area, there is no clear distinction in its implementation which can either be more tuned to software over hardware. The core ideology behind a research-oriented SDR is a system whose communication modules/algorithms can be easily modified or even completely re-engineered to test and prototype research ideas with faster turnaround times. The benefits of using a more software-oriented SDR for research is mainly the ease of implementation, but this approach typically cannot achieve real-time speeds compared to its hardware implementation. Examples of such software-based systems are WARPLab [1] and SORA [13] that allow the user to program the baseband physical (PHY) layer in software, and use radio frontends for transmission and reception. On the other hand, hardware implementations are very time consuming to implement and often are even more difficult to change or modify than a similar software-based implementation. For instance, the Scalable Communications Core baseband system on a chip (SoC) [8] can be used to implement baseband systems of different
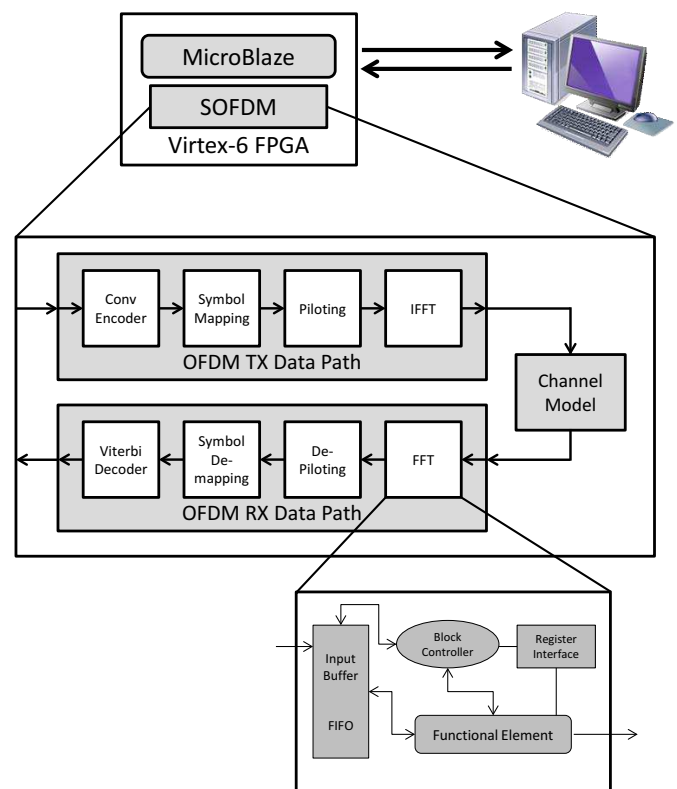

Fig. 1. The above image shows the modular breakdown of the entire system. The image only shows few of the implemented blocks and depicted to elaborate the overall system flow

standards with a network on chip (NoC) architecture [9] between functional modules implemented with coarse grained, heterogeneous and programmable accelerators. Such a system is purely aimed at achieving a scalable wireless baseband but lacks the capability for flexibility, outside the preassigned standards, for prototyping newer techniques involved with research platforms.

Where do we draw the balance between software and hardware implementation of the SDR? An ideal system would have the software-based flexibility and hardware-based speeds. The system must also be easily reconfigurable for research needs and focus has to be put into deciding where to build in this aspect of reconfigurability, either be on software side, the hardware side, or both. More importantly, the introduction

of reconfigurability comes at the price, in terms of added complexity and area/time overhead, of engineering the communication backbone necessary to control the overall system, as is the case with the technique introduced in AIRBLUE [11] for cross-layer communication.

This paper develops a programmable hardware implementation, called Scalable OFDM (SOFDM), of a generic digital baseband system for wireless protocols in a latency insensitive fashion that can be easily configured by a higher-level software layer and aims to facilitate research into SDR development, specifically cognitive communication systems based on orthogonal frequency division multiplexing (OFDM) schemes. Fig. 1 shows selected blocks comprising the overall system as well as the corresponding data flow. Our full implementation has all the relevant blocks needed to realize OFDM-based schemes, including channel estimation, carrier frequency offset, cyclic prefix, and packet detection blocks. The closest system to the one described in this paper, to our knowledge, is SDCR [7] which implements a baseband architecture on a smaller scale with the aim of building cognitive radio using dedicated hardware for the computationally intensive communication cores. Our system is another approach to defining a flexible radio where the hardware itself has room for flexibility through a software/data driven interface and therefore enabling research involving prototyping directly to hardware. The proposed hardware architecture supports flexibility and ease in maintaining its functional correctness even if the latencies of individual pipeline stages are changed, either at design time or at run time. We envision it to be used for rapid prototyping of different standards as well as run-time adaptation in both new and known OFDM schemes.

The paper is organized as follows. Section II discusses related work while Section III describes the key processing blocks needed for OFDM-based SDRs. We develop the latency insensitive architecture in Section IV and Section V presents some experimental results. We conclude the paper in Section VI with a discussion on future work.

Standard	Encoder rates	Modulation scheme	IFFT size
802.16 WiMAX	1/2, 2/3, 3/4, 5/6	BPSK, 4-QAM, 16-QAM, 64-QAM	128, 512, 1024, 2048
ECMA 368	1/2, 2/3, 3/4,	BPSK, DCM	128
802.11n WLAN	1/2, 2/3, 3/4, 5/6	BPSK, 4-QAM, 16-QAM, 64-QAM	64
802.11a WLAN	1/2, 2/3, 3/4	BPSK, 4-QAM, 16-QAM	64

TABLE I.   POSSIBLE CONFIGURATION SETTINGS FOR THE VARIOUS STANDARDS.

II.   RELATED WORK AND RESEARCH

A SDR platform mainly consists of two major layers apart from the applications itself, namely the media access control (MAC) and the physical (PHY) layers. The MAC layer consists of packing, unpacking, error checking and transporting data in a timely fashion to and from the communication baseband which resides on the PHY layer below with the application layer above it. While the MAC layer is a predominantly straightforward implementation, the PHY layer can be considered the holy grail for research. The PHY layer, better referred to as the baseband layer, is mainly divided into two areas in any SDR system: a hardware focused area and a software focused area. The hardware side mainly consists of antennas, A/Ds and D/As and the software side consists of three groups consisting of the filter stage, modem stage and the codec stage. The software side is highly volatile and therefore significantly different with respect to the different radio standards implemented as it works with code libraries customized specific to the needs before downloading processed data onto the board. The filter stage focuses on enforcing band limitation and is placed right before and after the D/A and A/D respectively. The modem stage is also called the inner transceiver. It is the most diverse amongst different standards and key in signal conditioning, which involves rake reception, correlation, synchronization, detection, equalization, FFT, OFDM, mapping, de-mapping, matrix multiplications, inversions, etc. This is the most intensively researched stage of SDR since it has room for algorithm evolution to improve throughput and performance leading to better performance within power constraints. The codec stage is called the outer transceiver and takes care of data manipulation outside the immediate frame/symbols of data which involves encoding, decoding, interleaving, de-interleaving and a variety of established channel algorithms like Turbo, Reed-Solomon and Viterbi. This stage, which is a heavily computational implementation of these kernels, makes it a good area to provide hardware support to take it off the critical path while implementing SDR.

A wide range of SDR platforms are available off the shelf but are limited in their use due to various bottlenecks involving host-platform interface speeds [2], insufficient processing power, and memory capacity. Platforms such as HYDRA [10] and Microsoft SORA [13], though popular for promoting SDR research, tie their users to predefined standards and speeds that do not ensure their longevity given the current nature and pace of SDR research. In particular, signal/data processing along with algorithm development for newer techniques inclusive of spectrum sensing, adaptive learning, SVD [14], etc in SDR research is gaining increasing importance. Implementing these newer techniques, however, requires flexibility within the hardware fabric, as well as increased processing power and memory/interface bandwidth—requirements not supported by existing platforms.

Researchers have developed platforms supporting OFDM-based standards, including the Wireless Open-Access Research Platform (WARP) [1], AirBlue [11], and Software-defined Cognitive Radio (SDCR) [7]. Though quite useful as a research platform, WARP supports the 802.11a/g standards whereas our architecture can be easily configured and flexible over a larger number of standards and variations within specific standards because of its standard independence at the modular level . Similar to our proposal, AirBlue introduces a latency-insensitive design as well for the OFDM pipeline but lacks the ability to configure the individual stages at run time, and therefore, cannot support a real-time adaptive OFDM PHY. The SDCR platform supports a limited form of an adaptive OFDM PHY in that only the modulation and FFT/IFFT stages are configurable at run time. Our SOFDM design allows all pipeline stages to be configured, including coding and interleaving.

Individual stages within our SOFDM pipeline have been

designed to achieve different data rates based on the configuration settings chosen from Table I [6] [5] [3] [4]. The overall pipeline, however, must remain insensitive to the latencies incurred by individual stages. We accomplish this goal by introducing buffers within each stage and designing the stages themselves to be stallable. Considering neighboring pipeline stages as producer and consumer, we define stallability as follows, the producer waits for the consumer to signal its readiness prior to transferring data; the consumer starts processing as soon as data is available in its local buffer; and the producer stalls when the consumer's buffer is full, which occurs if the production rate exceeds the consumption rate. We show how to design the appropriate synchronization/signaling scheme and implement the SOFDM pipeline on a Virtex-6 FPGA clocked at 100 MHz. The performance is characterized in terms of functional correctness (using waveforms) and the area cost. Our implementation currently supports coding rates of 1/2 and 3/4, the BPSK, 4-QAM and 16-QAM modulation schemes, and OFDM symbol sizes of 32, 64, 512 and 1024.

Our adaptive OFDM system follows the hierarchical structure shown in Fig. 1 consisting of the host PC connecting to the baseband implementation on a Virtex-6 FPGA with a MicroBlaze that functions as the central controller. Further down the hierarchy, we can see the latency insensitive architecture built into every module incorporating the baseband system.

III. OVERVIEW OF THE SCALABLE OFDM ARCHITECTURE

This section familiarizes the reader with a typical OFDM architecture [12]. The hardware components comprise of the antennas, A/Ds and D/As, whereas the programmable components can be categorized into three major groups: filters, modems and codecs. *Filters* focus on generating a band-limited signal and are placed right before and after the D/A and A/D stages. *Modem* performs signal conditioning which involves rake reception and correlation (to mitigate the effects of multipath fading), synchronization, detection, equalization, FFT, and OFDM. *Codec* stage is responsible for encoding and/or decoding the digital data using error-correction algorithms such as Turbo codes, Reed-Solomon, and Viterbi, as well as interleaving and deinterleaving. The key functional stages of the OFDM pipeline are explained below.

Encoder/Decoder: Given a set of input data, the convolution encoder computes the corresponding codeword using a specific generator polynomial (realized as XOR functions) and a decoder, such as the Viterbi, uses the same polynomial to predict the most likely sequence of bits received.

Interleaver/Deinterleaver: The interleaver permutes data across a block to improve the correction of burst errors and average power across the symbol. Interleaving can be done either via inter-symbol or intra-symbol permutation of data, that is either between data within a single frame or between data in adjacent frames. Similarly, the deinterleaver permutes the received data back in order, based on the vector used for interleaving at the transmission site.

Modulation/Demodulation: Modulation maps the input data to real and imaginary values for the IFFT block to be convolved in the time domain. Conversely demodulation maps the data received from the FFT block at the receiver back

into its constituent bits using a decision matrix tailored to the underlying modulation rate. It is important to note that the modulation/demodulation stage works on blocks of data to function and therefore alignment has to be strictly controlled during scaling.

IFFT/FFT: The OFDM baseband uses IFFTs at the transmitter to code the data into subcarriers and FFTs to decode the data off subcarriers at the receiver site. The OFDM symbol size varies based on the standard being implemented and so does the IFFT size. Another stage closely tied to the IFFT/FFT block is piloting—the process of reading OFDM frames and inserting known values into them as place markers for the receiver to use to align incoming frames. The piloting stage must create well-aligned frames for the IFFT/FFT block to process in a timely fashion.

Finally, as noted in the Introduction, a typical hardware based physical layer is not flexible since the pipeline stages and associated control logic are tailored for specific standards. Any change in the latency incurred by a stage due an attempt to modify a module affects the rate, which, in turn, results in incorrect pipeline operation. Other significant components that were built for this project are as follows.

Zero Padding: It is important to have the functionality to fit the entire input data into OFDM symbols. Since the working frame size or blocks of data that traverse our system is reconfigurable, a dynamic zero padder is necessary, where its parameter is set using a single register. It is vital to have the same parameter on the receiving end to ensure a working depadder, which can be achieved either by transferring or by calculating the value on the receiver. Padding done before the encoder is not advisable since encoding dummy data inserted for padding purposes would be waste of resources.

PLCB Header: Different standards have different header sizes. The header module has been flexibly built to be able to send a variable set of data right before the data payload. In order to have a scalable receiver, the time required to set the receiver chain will be quantized and the received data would also be delayed and buffered till the receiver chain is ready. The PLCB header is often sent using BPSK to ensure the lowest error rates at the receiver even at high noise levels.

Piloting: The system has configurable parameters corresponding to the pilot position, the pilot value, and the symbol size. The flexibility developed into the piloting controller is used to allow the implementation of *non-contiguous OFDM* by allowing specific subcarriers to be nulled during processing.

Non-Contiguous OFDM (NC-OFDM): The system was built with provision for selective loading of the subcarriers and thereby capable of implementing NC-OFDM. This module is designed so that the mapping vector can be modified (with any desired value or null) for intelligent loading of subcarriers based off a spectrum sensing module connected to this model's controller in the future.

Packet Framing: This is a vital part of the scalability of our system, where various encoding, modulation rates and IFFT sizes are supported. The system was build to compute and calculate the various parameters based on initial conditions set by the user for the system and then set every corresponding control logic upon the set conditions automatically. At the

Pipeline stage	Configurable parameters
Encoder	Coding rate, polynomial
Interleaver	Inter symbol and Intra symbol interleaving
Modulation	Modulation scheme, data mapping value
Piloting	Pilot position, pilot value, symbol size
IFFT	Symbol size, guard prefix

TABLE II.   THE VARIOUS CONFIGURABLE PARAMETERS SUPPORTED
BY THE PROPOSED SOFDM SYSTEM.

transmitter the data being produced is held in memory until either a significant chunk or complete data is available for transmission. Buffer sizing for cases where the user chooses to send huge payloads therefore would have limitations based on resources available for that particular setting.

Packet Detection: Packet detection is a computationally expensive procedure at the receiver. It can be based either on received average power or cross correlation of received data with a known packet detection sequence. In this model we built a multi point tap correlator to detect packet start, keeping in mind the strict synchronization requirements to prevent missed detections. The detector was made with simple arithmetic without up-sampling data in order to keep the entire system's clock rate high at the expense of logic utilization on the FPGA.

IV. Latency-Insensitive Pipeline

The SOFDM pipeline accommodates design time and run time changes to the configuration parameters for the codec, modulation, piloting, and IFFT stages. Table II lists few of the various parameters of interest specific to each stage.

To seamlessly support the parameters listed in Table II, the pipeline is designed to be insensitive to the latencies incurred by individual stages. This is achieved by designing stages to be stallable as shown in Fig. 2. Here we choose a simple scheme wherein data is propagated among stages along with a valid signal and a control bit that differentiates the header from the payload. Pipe stages are managed by a global controller to initiate packet processing using the `PKT Start` and `Global Enable` signals, and bypass the functional processing path using the `Bypass` signal if necessary. The bypass function enables faster debugging, allowing end-users to skip the functional processing path and simply pass on the input data to the following stage and pinpointing the errors through isolation. Each stage uses the generic architecture shown in Fig. 2 and a brief discussion on each of the pipeline stages follows.

Input Buffer: This buffer is realized as a FIFO that latches data from the previous pipeline stage on an active `Data Valid` signal. This buffer eliminates the functional dependency between pipe stages; each stage processes the available data at its input and passes it on to the next stage. Thus, the processing latency of a stage is independent of other stages in the pipeline, assuming that data is available in the input buffer. The buffer size at any stage is set to twice the number of data items required to form an OFDM symbol.

Register Interface: The interface allows the software layer to configure the SOFDM system in response to the prevailing channel condition and data rate requirements. The interface is implemented as a shared register block and is currently

managed via an in-house MATLAB code and through code running on the MicroBlaze. The register values are updated upon a write request without any latency and the shadow copy of the register inside Functional Element is updated as per the chosen polling scheme.

Block Controller: It manages the read datapath of Input Buffer and generates the data format required by Functional Element. The processing path of Functional Element is disabled until enough data points are available in the buffer and the "ready for data" (or `RFD`) signal is asserted by the next stage. This block also monitors the buffer size and asserts the `RFD` to the previous stage. So, the `RFD` serves as a stall indicator to the previous stage of the pipeline, thereby avoiding a buffer overflow.

Functional Element: This block processes the input data and its functionality changes with each stage of the SOFDM pipeline. For example, Functional Element in the encoder stage performs convolution encoding supporting the data rates listed in Table II and similarly this element within the modulation stage performs symbol mapping. The data path and the control path within Functional Element are stand-alone and can be replaced with new functionality as along as the signals generated by the above components are respected.

Apart from the major components discussed above, another relevant factor to take into account is the buffer sizing. As we discussed earlier, the SOFDM system is highly configurable to support multiple standards. Different standards have different throughputs depending on supported data rates. This would need a change in the processing logic as well as data buffering logic of the OFDM pipeline. Buffer sizes, which could sustain throughput of one configuration, might not be compatible for another configuration, thus resulting in throughput losses necessitating a methodology to choose accurate buffer sizes. This would also allow us to reduce the area and power of the generated hardware. The buffer sizes at each stage of the OFDM pipeline are governed by buffer size equations shown in Table III.

V. Experimental Setup and Results

The SOFDM system was built using Xilinx System Generator; a plug-in for MATLAB's Simulink framework. Additionally, Verilog was used to create custom hardware controllers and elements that were not readily available from System Generator. After development with System Generator, the SOFDM model was synthesized for a Xilinx ML605 development board for the Virtex-6 FPGA. In the co-simulation setting, the design uses MATLAB's point-to-point Ethernet flow to send data from MATLAB running on the host PC to the board and received data back into MATLAB's workspace for running statistical comparisons on bit error rates. The main drawback to using MATLAB's co-simulation framework is the fact that the connection speed is significantly low. However, this method is the easiest to implement on the fly as flashing the board with the changes is taken care of by MATLAB tool chain. There are two other methods to connect to the board that uses the Gigabit Ethernet connection and the PCIe connection on the ML605. The latter two connectivity options are much faster and involves the transfer of our generated design into Xilinx's EDK environment and run controlled through MicroBlaze.
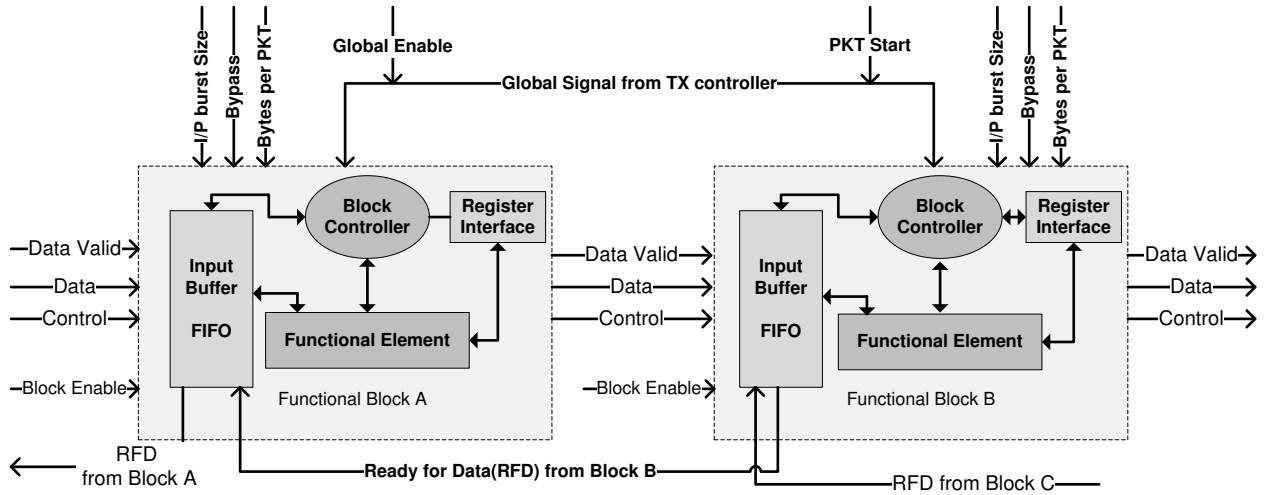
Fig. 2. The design of a stallable pipeline stage showing the data flow as well as the control signals exchanged between neighboring stages for synchronization purposes. The configuration parameters within a stage are also set by these control signals.

Pipe stage	Buffer Size Equation	Parameter Description	
Encoder/Decoder	$ENC_{BS} = MOD_{BS} \times CODE_{RATE}$	ENC_{BS}	: Encoder/Decoder Buffer Size
		$CODE_{RATE}$	: Encoder coding rate
Modulation/Demodulation	$MOD_{BS} = PILOT_{BS} \times MOD_{RATE}$	MOD_{BS}	: Modulation/Demodulation Buffer Size
		MOD_{RATE}	: Mapping rate, bits per mapped data point
Pilot/De-pilot	$PILOT_{BS} = (N_{SDP}) \times L$	$PILOT_{BS}$	: Pilot/De-pilot Buffer Size
IFFT/FFT	$IFFT_{BS} = (N_{SDP} + N_{SO}) \times L$	$IFFT_{BS}$	: IFFT/FFT Buffer Size
		N_{SDP}	: Number of Data Subcarriers
		N_{SO}	: Other Subcarriers (Pilots, Nulls)
		L	: Number of Symbols to be buffered
			for sustained throughput

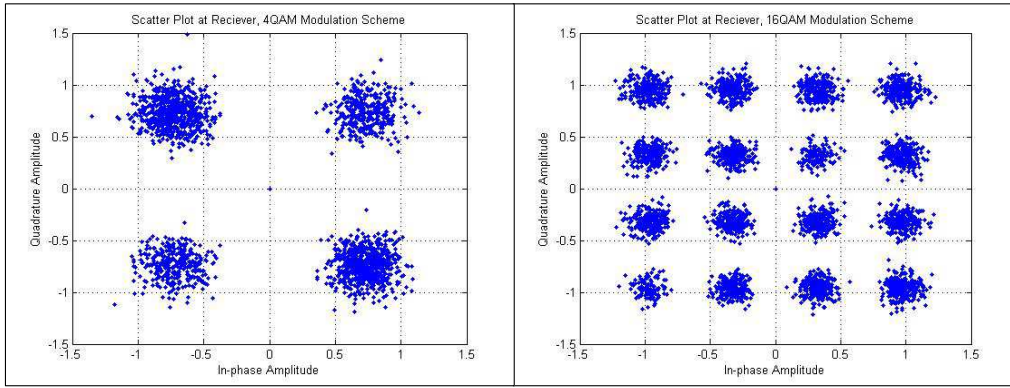TABLE III.   EQUATIONS FOR DETERMINING BUFFER SIZES WITHIN EACH OF THE PIPELINE STAGES.

Fig. 3. Constellation mappings corresponding to the 4QAM and 16QAM modulation schemes as seen at the receiver through an AWGN channel built on the Virtex-6 FPGA.

We have been using the co-simulation framework for quick simulation and the Gigabit ethernet for longer runs while the PCIe interface is still under development. The results in this section are generated using the co-simulation flow to capture modular control data and verify our designs. This design verification process shows the significance of our design as a wireless research platform where modules and algorithms can be added, tweaked or deleted with relative ease having granular access to every working element.

Each stage of the SOFDM system was separately built and tested for its correctness and configurability before being integrated into the final SOFDM system. The module development flow involved the following steps:

- *Code Implementation* using MATLAB scripts

- *Simulink Implementation*, where the blueprints are laid

- *System Generator Implementation*, where the module controllers in Verilog/VHDL are written and tested. This step takes the longest time.

- *SOFDM Testbed* is then created to run all the implemented modules together.

For validating the data transmission process we send a greyscale picture and numbers from a free running counter
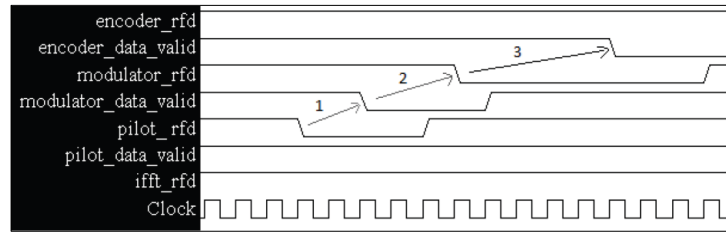
Fig. 4. Waveform showing how a stall caused by the piloting block propagates backwards through the various pipeline stages. The propagation is indicated by a sequence of numbered arrows.

and crosschecked the results. SOFDM implementation also has packet detection, channel estimation, frequency/phase correction, scrambler, equalizer, scaling blocks with other fine tuning blocks integrated into the model. The programmable registers are globally located in the design and are currently either changed manually before running or populated through startup scripts from MATLAB after which its controlled through MicroBlaze based on user inputs either during initialization or at runtime.

Resource Name	Count	% Utilization
LUT's	18645	12
FF's	15891	5
BRAM's	37	8
Mult/DSP48	29	3

TABLE IV.   RESOURCE UTILIZATION OF THE SOFDM SYSTEM ON THE VIRTEX-6 LX240T FPGA.

Fig. 4 shows the synchronization signals exchanged between the various stages of the SOFDM pipeline in terms of the RFD and Data Valid signals. The waveform shows that when the piloting block pulls down the pilot_rfd signal indicating that it can no longer accept input data, the modulator block acknowledges this by pulling down the modulator_data_valid signal, thereby not sending new data. This stalling affect is propagated through the entire pipeline. In order to capture control data on the host PC, the design is clocked at 100 MHz, where the limitation is imposed by the co-simulation interface. However, the place-and-route timing analysis on just the SOFDM pipeline indicates that it is capable of running at 150 MHz. This validates the functional correctness of our design and also opens new realms where the introduction of flexible hardware can be significantly used in researching newer areas like bit-loading and cognitive radios. Finally, Table V lists the logic utilization of the mapped design on the Virtex-6 LX240T FPGA. The area is significant as it only utilizes less than 25% of the FPGA leaving area on the fabric for future research applications.

VI. CONCLUSIONS AND FUTURE WORK

We have developed an SOFDM pipeline capable of seamlessly accommodating different standards and different configuration settings for a single standard. The pipeline architecture is designed to be insensitive to the latencies incurred by its individual stages. We have developed and validated the synchronization scheme and demonstrated an implementation of the pipeline on a Virtex-6 LX240T FPGA. It currently supports multiple coding rates, modulation schemes, and OFDM symbol sizes, with the provision to scale further. Also, the system's flexibility in terms of selective loading of the subcarriers makes it a suitable platform for research in adaptive spectrum sensing and cognitive communication. In future work, we will further reduce the end-to-end pipeline latency by optimizing the input buffer sizes within each stage, and by accelerating the piloting and decoder blocks.

REFERENCES

[1] "Rice University WARP - Wireless Open-Access Research Platform (WARP)." http://warp.rice.edu.

[2] GNU Radio Overview. http://gnuradio.org/redmine/projects/gnuradio.

[3] "ECMA-368, Standard:High rate ultra wideband PHY and MAC standard, 3rd Edition, December 2008.".

[4] "MBOA standard, MultiBand OFDM Physical Layer Proposal for IEEE 802.15.3a, September 2004." http://www.wimedia.org/imwp/idms/.

[5] IEEE 802.16-2009 standard for local & metropolitan area networks part 16: Air interface for broadband wireless access systems. 2009.

[6] IEEE 802.11 standard for wireless lan medium access control (mac) & physical layer (phy) specifications, 2012.

[7] A. Dutta, D. Saha, D. Grunwald, and D. Sicker. An architecture for software defined cognitive radio. In *Symp. Architectures for Networking & Communications Syst. (ANCS)*, pages 1–12, oct. 2010.

[8] J. Hoffman, D. Ilitzky, A. Chun, and A. Chapyzhenka. Architecture of the scalable communications core. In *Proc. Symp. Networks-on-Chip*, pages 40–52, may 2007.

[9] P. Jing-wei and Y. Bo. A new fabric of dynamic noc for communication in reconfigurable devices. In *Proc. Conf. Computer Technology & Development*, pages 590–593, nov. 2009.

[10] K. Mandke et al. Early results on Hydra: A flexible MAC/PHY multihop testbed. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 1896–1900, 2007.

[11] M.-C. Ng et al. Airblue: A system for cross-layer wireless protocol development. In *Symp. Architectures for Networking & Communications Syst. (ANCS)*, pages 1–11, oct. 2010.

[12] H. Schulze and C. Lueders. *Theory and Applications of OFDM and CDMA: Wideband Wireless Communications*. Wiley, Hoboken, NJ, 2005.

[13] K. Tan et al. Sora: high performance software radio using general purpose multi-core processors. In *Proc. USENIX Symp. Networked Systems Design & Implementation (NSDI)*, pages 75–90, 2009.

[14] Y. Wang, K. Cunningham, P. Nagvajara, and J. Johnson. Singular value decomposition hardware for mimo: State of the art and custom design. In *Proc. Conf. Reconfigurable Computing & FPGAs*, pages 400–405, 2010.