A MULTI-MODE RECONFIGURABLE OFDM COMMUNICATION SYSTEM ON FPGA

Qingbo Wang, Ling Zhuo, Viktor K. Prasanna

Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
{qingbow, lzhuo, prasanna}@usc.edu

John Leon

Irvine Sensors Corporation
Costa Mesa, CA 92626-4526
jleon@irvine-sensors.com

ABSTRACT

Wireless communication channels suffer a variety of environmental interferences. Thus, it is imperative for mission critical communication systems to have the ability to adapt to run-time channel conditions. An adaptive wireless communication system can be made multi-modal by implementing various modulation or coding schemes, or by transitioning between different communication technologies, such as SISO-OFDM and MIMO-OFDM.

In this paper, we present an FPGA-based multi-mode reconfigurable OFDM wireless communication system. The proposed design enables the system to switch between different communication modes, e.g. SISO-OFDM with QPSK or QAM-16 as modulation schemes. The system is composed of modular functional blocks, such as handshake protocol, channel measurement trigger, transmission error rate measurement and mode switching and recovery. It also includes schemes to enhance packet delivery on low quality wireless links and reduce data loss during the mode switching process. The implementation utilizes one PowerPC, 75% of the slices and 80% of the BRAMs on a Xilinx Virtex Pro II VP70.

1. INTRODUCTION

Currently, advanced wireless communications can achieve a very high data rate with Orthogonal Frequency Division Multiplexing (OFDM) technology. However, wireless communication is usually less reliable than wired communication, and wireless communication channels suffer more from environmental interferences. Thus it is desirable for communication systems –especially for mission critical systems– to be able to detect environmental changes and switch the communication mode based on channel conditions and system performance objectives.

The IEEE standard 802.11a [1] has various communication modes with possible data rates of 6, 9, 12, 18, 24, 36, 48, and 54 Mbits/s. For a communication system to adapt fully to different operating conditions and standards, there must be not only the real time transition of modes within a wireless communication protocol, but also a transition between different communication technologies, such as SISO-OFDM and MIMO-OFDM.

FPGAs have long been considered an attractive option for high-performance implementations of wireless communication systems [2]. The demand for adaptability on such systems can be satisfied by currently available software and hardware techniques on FPGAs. Leading field programmable device vendors provide technologies to change the logic function of application systems dynamically, thus enabling the design of adaptive communication systems.

However, when realizing a run-time mode switching on such adaptive systems, many issues must be addressed. For example, we must design the triggers to start channel measurements, determine measurement metrics, select mode switching mechanisms, and ensure that packets can be transmitted reliably even in wireless channels with poor connectivity. Moreover, a coordination protocol is needed to integrate all these individual designs for the system to work with multiple wireless nodes.

We propose a modular design to facilitate the mode switching between communication schemes. By decomposing the system into multiple functional modules, we exploit the modularity inherent in these systems. The system design for mode switching includes handshake protocol, channel measurement trigger, transmission error rate measurement, mode switching and recovery, as well as schemes to enhance packet delivery on a low quality wireless link and to reduce data loss due to mode switching. These mechanisms, except for the handshake protocol, work as individual actions at different stages of the handshake protocol procedure.

The modularity of our implementation allows individual modules to be substituted for others. For example, the measurement of transmission error rate can be purely software-based or hardware-generated BER (bit error rate), depending on the availability of certain functional features on the

This research is supported by grant DOD-FA9550-05-C-0183.

development platform. The mode switching module can be implemented either by reconfiguring the hardware logic or by changing control parameters to adjust the functionality of the module. In addition, the trigger functions can be manually and/or automatically controlled, and can be turned on and off based on user demand. We also propose a buffer management mechanism to alleviate data loss during mode switching. The proposed design is modular enough to accommodate this diversity.

Our experimental system is based on the WARP [3] hardware platform and utilizes its wireless open-access research framework. The development boards include one FPGA board with Xilinx Virtex II Pro, and several add-ons, such as a radio board and a clock board. During our experiment in lab environment, the completed system achieved wireless bandwidth from 2 to 7 Mbps depending on the different modulation schemes. The system shows robustness and takes about 200-560 ms to finish the handshake protocol and switch from one mode to another. This time depends on the error measurement scheme and the timer setups for automatic packet re-transmission needed by the handshake protocol. The implementation utilized one PowerPC, 75% of the slices and 80% of the BRAMs on the Xilinx Virtex Pro II VP70.

In Section 2, we present related work. We introduce our design in Section 3. In Section 4, we present our hardware implementation on FPGAs. The experimental setup and the results are discussed in Section 5. In Section 6, we conclude by discussing opportunities for improvement.

2. BACKGROUND AND RELATED WORK

OFDM has become standard technology to improve channel utilization for wireless communication. Frequency Division Multiplexing (FDM) transmits multiple signals simultaneously over a single path. Each signal has a unique frequency range (carrier). Orthogonal FDM (OFDM) [4] is a special case of FDM where a single data stream is distributed over several lower rate sub-carriers. In other words, one signal is transmitted by multiple carriers. Sub-carriers are separated by given frequency ranges to avoid cross-carrier interference. The benefit of orthogonality is a high spectral density. A third dimension, space, is introduced into the traditional frequency-time domain. The idea is to transmit multiple streams of data on multiple antennas at the same frequency to increase throughput. Typically, multiple receiver antennas are used as well. Multiplied by the number of channels between either end, this configuration achieves high data rates. This principle is called Multiple Input Multiple Output (MIMO) [5].

Many efforts have been carried out on the implementation of SISO- and MIMO-OFDM physical layers from research areas involving DSP, VLSI and FPGA. An OFDM Wireless Transceiver was implemented using IP Cores on an FPGA [6]. In [7], Masselos K, et al. discussed the implementation of wireless multimedia communication systems on reconfigurable platforms.

FPGA-based designs have performance advantages over DSP designs of OFDM communication systems. For example, the design discussed in [8] aimed to implement a wireless communication physical layer compliant with the 802.11n standard. The authors identified the pipeline property in the architectural design of such streaming systems, and used one streaming FFTx IP core to fulfill the demand for two high speed FFT computations, thus reducing the cost of hardware development.

Researchers have explored applications invovling dynamic reconfiguration on FPGAs. In [9], the authors proposed a way to convert a bus into a group of point-to-point connection harnesses, which can facilitate the reconfiguration process. Fu, et al [10] studied scheduling algorithms for reconfiguration, especially performance optimization through adjusting the scheduling intervals between each configuration. An application utilizing the hardware dynamic reconfiguration technology in signal processing was introduced in [11].

In our study, we focus on the general procedure for an adaptive wireless communication system to conduct mode switching based on measured channel conditions. We implemented a system with such capabilities on a commercially available FPGA development platform [3].

3. RUN-TIME MODE SWITCHING SYSTEM

In this section, we outline our system design, and describe the functionalities of individual modules, as well as the interaction between these modules. A handshake protocol acts as the skeleton for the entire system and coordinates other components. We start by presenting a system overview from the perspectives of participating wireless nodes.

3.1. Node Operational Flow

In this paper, a mode switching takes place between two participating wireless nodes, one designated as a server, and the other as a client. However, the scheme can be easily adapted to more complex scenarios with one server and multiple clients. Note that we use "mode switching" at times to refer the entire mode switching process (or protocol), which is from the beginning of a trigger signal to the completion of an actual communication mode switching.

As illustrated in Figure 1, there are two ways to initiate a mode switching. The first is to adopt an automatic trigger signal that is generated periodically after the system powers up. The second way is to push a button or flip a switch manually to generate a trigger signal. These signals cause interrupts to the system. The interrupt service routines (ISR) then start the handshake protocol, and turn the node into a mode

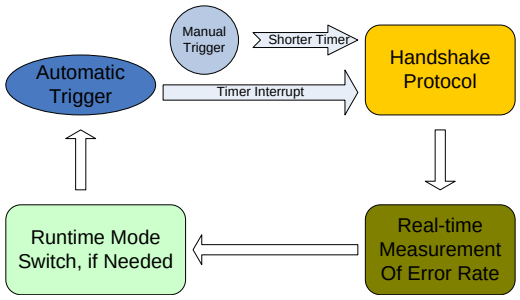

Fig. 1. System Overview

switching state. The mode switching state is a state where the nodes conduct mode switching protocol only, contrary to the normal state, where normal data communication between the nodes takes place. Only server nodes have this trigger mechanism to initiate the mode switching process. After starting the handshake protocol for mode switching, The server node contacts its client, which then also enters mode switching state. The client side operations are different from those on the server side, and both are described in detail in the next section. Through the exchange of the handshake protocol packages, the server and its client reach an agreement on when to start measuring the channel conditions. The results of measurement are an indication of the channel quality, and used to decide whether a mode switching is needed. Measurement schemes are selected based on preferred channel quality metrics and available facilities in the software and hardware environment for system design. This scheme should be agreed upon beforehand between server and client so that all participating nodes interpret the measured results in a consistent way.

In our design, the two trigger schemes both function in run-time. They use the same timer to generate interrupts. However, the manual scheme sets a shorter expiration time for the timer, so that the handshaking appears to start immediately after the button is pressed.

3.2. Handshake Protocol

The handshake protocol works in three phases, as illustrated in Figure 2. The first phase is called the "initiation phase." When interrupted by an expired timer, the mode switching server stops all routine activities and enters the mode switching state. The server sends "start" packets to the client periodically, until it receives an "ack_start" from the client. On the client side, the "ack_start" is sent out periodically upon receiving a "start" packet. This phase is completed when the server receives the "ack_start." By that time, both parties confirm they all enter the mode switching state and are ready for the next phase.

The second phase is called the "measurement phase."

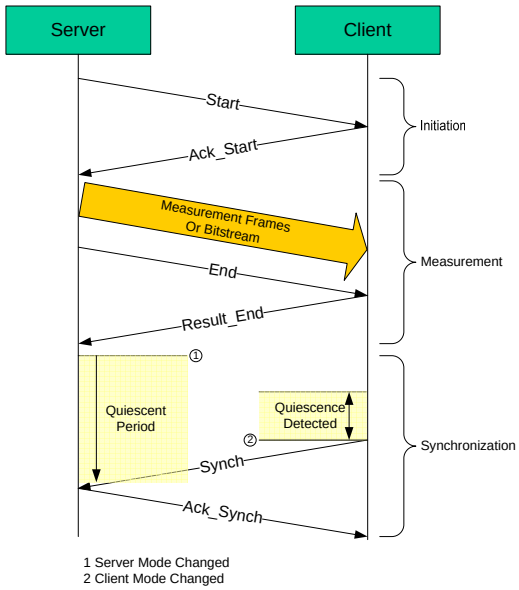

Fig. 2. Illustration of the Handshake Protocol

The server continuously sends a predefined bitstream, or a sequence of predefined short packets. After all the data is transmitted on the server side, the server sends "end" packets to inform the client that no more "measurement" packets will be sent. Once receiving an "end" packet, the client calculates the transmission error rate, and then sends the result back in a "result_end" packet. We will explain the transmission error rate and its calculation in Section 3.4. This "result_end" packet is equivalent to an acknowledgment packet to the server. Receiving the "result_end" notifies the server that the measurement succeeded, and concludes this phase.

In the last phase, the "synch phase," the nodes change the communication mode based on the measured result, and then go back to the normal state for communication. The mode switching decision is made by individual nodes based on a pre-defined mode switching table. All participating nodes store the same table in their memory. Note that at the beginning of this phase, the client is still sending the "result_end" packets, without knowing whether the server received this acknowledgment. When the server receives the result, it may initiate a mode switching according to the pre-defined table. However, this uni-lateral action leaves the server and client nodes working in different modes. Hence the server can no longer communicate with the client node, and the client does not know whether it should change modes.

We devised a "quiescence" scheme to handle this situation: After switching its mode, the server stops any transmissions over the air. The client sets up a "quiescence" timer while continuing to send "result_end" packets to the server side. This timer is reset to the initial state whenever an valid "end" packet from the server side is received. When this

timer expires, it means that the client has not received any "end" packets from the server in a given time period. This informs the client that the server has already changed its mode. At this moment, the client is safe to change its mode according to the same mode switching table. After that, the client sends out "synch" packets periodically. The server acknowledges those packets with an "ack_synch." This exchange leads both nodes to return to the normal state.

3.3. Mode Switching

The mode switching mechanism is a critical function of this system. There are two ways to switch communication modes based on the current available technologies. One is to build a comprehensive core, which includes the necessary functionality for all the modes. By design, different mode operations share a certain amount of hardware logic within the core. To allow different logic blocks to collaborate with each other, special control logic is implemented to (re)arrange all blocks to achieve requested mode functionalities, thus realizing mode switching. This is similar to the Software Programmable Reconfiguration approach (SPR) [12], except that the mode switching control is achieved via custom-designed logic. The other way is to follow the Dynamic Partial Reconfiguration design flow (DPR) [13]. This approach requires multiple designs for each reconfigurable subregion inside a core. These different designs are compiled into corresponding partial reconfigurable bitstreams, all of which are stored in flash memory. When it is time for reconfiguration, the PowerPC on FPGA reads in the appropriate partial design bitstream from flash memory, and writes it to the FPGA configuration memory through ICAP. ICAP is Internal Configuration Access Port, through which PowerPCs can access the FPGA configuration memory at run time. This bitstream re-wires the hardware logic in the reconfigurable subregion on the FPGA. The first method takes less time to finish functional alterations, while the second is area efficient comparatively. Our design adopts the first approach.

We use two different wireless communication modes for our example design - SISO-OFDM with QPSK modulation and SISO-OFDM with QAM-16 modulation. A SISO-OFDM baseband transceiver with QPSK modulation yields lower bandwidth, compared to a QAM-16 modulated SISO-OFDM transceiver. However, it produces fewer packets with errors under the same channel condition, thus appearing more robust than the mode with the QAM-16 modulation. In our system, we employ a scheme to measure how many erroneous packets or bits are received over a fixed amount of data. The results of this measurement are used as an indication of channel quality to trigger switching between modes.

Figure 3 illustrates the mode switching scheme used by our system. In this design, SISO-OFDM with QPSK modulation switches to SISO-OFDM with QAM modulation, a more erroneous mode, to procure higher bandwidth when

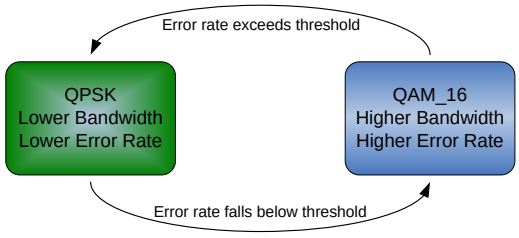

Fig. 3. Mode Switching Strategy

the nodes find that the channel condition is in good quality. Similarly, SISO-OFDM with QAM-16 can switch back to the less erroneous mode with QPSK modulation to reduce error rate in transmission when the channel quality worsens.

3.4. Key algorithms and Design Issues

The design for measuring transmission error rate involves two possible methods. One uses a software method only. A large number of pre-defined short packets are sent from the server to the client. On the client side, the received "good" packets are counted. At the end, the total number of the good packets are sent back to the originating server. This measured result is then used to decide possible mode switching actions. The second method utilizes the built-in hardware bit error rate (BER) generation unit in the WARP hardware platform. A pre-defined bitstream is sent from the server to the client. The client, loaded with the same bitstream, can compare the received bitstream with the one stored in its memory to count the number of erroneous bits, thus obtaining the BER. This result is also sent back to the server for decision making on mode switching. This hardware-assisted function may not be available in other development environments, so our current design uses the software method. Note that the results received at server side are unlikely wrong, due to the use of CRC ckecksum in the packet header and the re-transmission mechanism employed by the handshake protocol.

This system design must consider the unreliable wireless connection at the time of mode switching. Mode switching typically occurs when the link quality fluctuates, usually worsening. A common way to obtain reliability between two nodes is to retransmit packets automatically. This scheme was first adopted in our design. The server starts a timer when it sends out a packet. If the server does not get an "ACK" when the timer generates an interrupt, it will retransmit the last packet. However, it is possible the ACK is lost or corrupted over the transmission. Moreover, on our experimental platform, the original packets and their acknowledgements often collide with each other. Since the entire mode switching process must complete quickly, we can not afford re-transmissions than necessary. An better

ACK transmission scheme can improve the performance of the handshake protocol. As shown in Figure 4, besides the

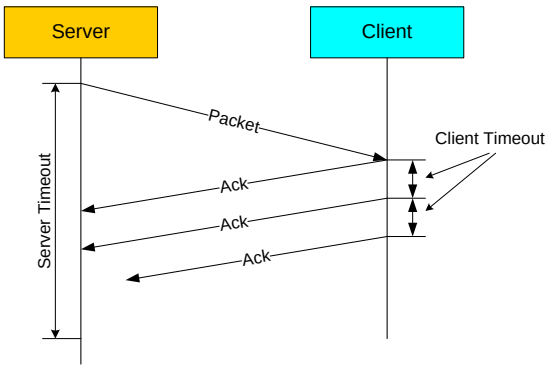

Fig. 4. 2-timer Scheme for Reliable Transmission

timer used on the server side for the original packets, we use another timer with a shorter expiration time set up on the client side for the ACK packets. That is, both the server and client implement a re-transmission mechanism. This scheme increases the probability that ACKs are received, and can circumvent possible collisions on our hardware platform to improve the ACK delivery. Our run-time experiments proved that the scheme works reliably and achieves better performance for the handshake protocol, thanks to fewer re-transmission overhead.

To mitigate data loss over the entire mode switching process, we introduce an SRAM buffer to store incoming packets before sending them for processing. The size of the buffer depends on the application's incoming data rate and the time spent on the entire mode switching process. Overflow may occur if the incoming traffic exceeds a data rate threshold, at which the SRAM buffer size is deducted. Also, it is possible, although highly unlikely, that the server and client mis-communicate and switch to different modes. We must enlist an exception handler to prevent the nodes from losing contact with each other. In our design, we use a "rollback" mechanism. When transmitting non-measurement protocol packets, the nodes use base modulation (BPSK in our system) for the packet header transmission. Thus, a node always knows the source node from which the packets are arriving by looking at the packet header's source address. The node can tally the number of packets it receives from certain node, but whose payloads can not be decoded. When this number exceeds a preset threshold, the node concludes the communication modes between the two are asynchronous. This node then sends "rollback" packets using base modulation to the source node, which acknowledges the rollback packets using "ack_rollbacks." The two nodes recover to their last viable, common communication mode, and resume the handshake protocol operation.

4. SYSTEM IMPLEMENTATION

4.1. Implementation platform

We used WARP boards [3] as our implementation platform. This toolkit is a scalable and expendable development environment for advanced wireless networks. We used the FPGA, Radio, and Clock boards. The FPGA board is the mother board with a Xilinx Virtex II Pro VP70, 4MB of SRAM, and extension slots. It is used to implement all the PHY layer logic, basic MAC layer protocol, and user level MAC protocols. Radio and Clock boards are add-ons, which are plugged into the extension slots on the FPGA board. The radio board implements the analog functionalities, including D/A, A/D, and RF transceiver. One can install 1 - 4 radio boards onto the FPGA board's extension slots to enable SISO or MIMO transmission. The Clock board is used to synchronize the clocks between different boards.

4.2. Implementation Details

Based on the basic WARP library, we developed a C program on PowerPC to extend the basic MAC layer protocol with our mode switching design. The trigger mechanism, handshake protocol, transmission error rate measurement, and mode switching control were all implemented in the category of "User-level MAC," as shown in Figure 5. Through the drivers and bus controllers, the MAC layer software controlled the underlying hardware logic and the access to SRAM on the board.

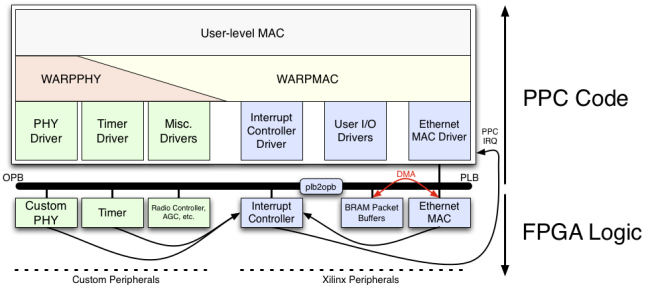

Fig. 5. System Diagram [3]

The timers used in this design were all implemented on hardware, and were controlled as peripherals by the software through the OPB bus. When firing, they register an interrupt to CPU, which then executes the corresponding interrupt service routine (ISR).

Part of on-chip BRAM was used as the instruction and data memory for the PowerPC processor. The other part was configured to serve as a buffer and store Ethernet and wireless packets, both inbound and outbound.

There are two banks of SRAM on the board, each of which is 2 MB. The first bank of SRAM was used as the

heap and stack for the PowerPC, while the second bank of SRAM was used to implement the buffer for data loss relief.

4.3. Packet Format

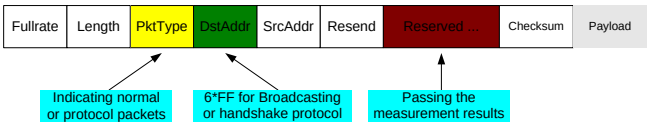

Fig. 6. MAC Packet Format with Definition of Special Fields

In this system, we utilized the MAC packet header format in the WARP development environment to facilitate the design of our protocol packets. From Figure 6, we see that a 64-byte packet header was divided into 8 fields. The MAC layer protocol checked the destination address of every incoming packet. We used the broadcast destination address of all '1' to specify that the packet is probably used for mode switching protocol. The packet header was further checked on its "pktType" field to determine which type the packet is. From Table 1, we see that packets were of two major types, one for normal data communication, including "Data" and "Ack." The others were for mode switch protocol use, i.e. the packets with pktType number higher than 1. At present, we have 9 kinds of different protocol packets. This "pktType" field is one byte long, so the maximum allowable number for different packets is 256.

Table 1. Packet Classification

Field number	Packet Type
0	Data
1	Ack
2	Start
3	Ack_Start
4	Measurement
5	End
6	Result_End
7	Synch
8	Ack_Synch
9	Rollback
10	Ack_Rollback

5. EXPERIMENTAL RESULTS

5.1. Experimental Setup

We used two PCs as the experimental stations. They each were connected to an FPGA board through its Ethernet port. The two FPGA boards were configured as a client and server by software. On one of the PCs (either the server or client), VLC media player [14] waited on a UDP port for incoming video packets. On the other PC, VLC was configured to send video clips to the waiting machine. In Figure 7, the video sender transmits the data, while the video receiver gets the data and plays it if the data is decoded correctly. The wireless link between the two nodes acted as a bridge only, transparent to both PCs.

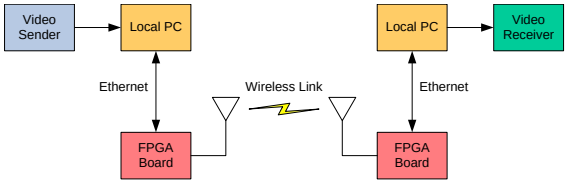

Fig. 7. Experimental Configuration

Replacing the VLC player with iperf [15] in the same setup, we measured the maximum bandwidth in terms of either TCP or UDP between two end points.

5.2. Experimental results

We conducted mode switching experiments in a lab environment between two SISO-OFDM wireless communication nodes, but with different modulation schemes, e.g. QPSK or QAM-16. Using the above setup and VLC player, video clips were streamed from one PC to the other for playing. The mode switching protocol ran periodically, and printed transmission error rate measurement results and current communication modes onto the text terminal interfaces via serial port every time a measurement was performed. A mode switching took place when the measured results for wireless link quality crossed a threshold. For the sake of experiment, we used a hair dryer in the vicinity of the wireless boards to introduce interferences. The generated magnetic noise led to changes in measurement of transmission error rate. Upon completion of a mode switching, new communication modes were displayed by the user terminal interface. When a mode switching was observed, the video display on the receiving PC showed little or no blocking effects, which means the time spent on the entire mode switching process was tolerable for the real-time video streaming on our experimental platform.

The wireless boards are shown in Figure 8. Without adding interference, we measured the available end-to-end bandwidth for the two modulation schemes using "iperf." The recorded measurement results showed variation between 2 to 7 Mbps.

The completed design utilized one PowerPC and 70% of FPGA slices. The BRAM usage is 264 out of 328 at about

Fig. 8. Experimental Boards

80%. The hardware utilization for each individual component can be found in Table 2.

Table 2. Utilization for Major Components

Component	Slice	Slice FF	4-input LUT
OFDM	8246	11241	10380
AGC	2535	3245	2631
Radio Controller	814	659	1222
Ethernet MAC	2532	2919	4085
Opb_timer	262	313	270
Ofdm_timer	117	178	141
PktDetector	406	468	430

During the experiment, the measured time for the entire mode switching process varied between 0.5 and 0.6 seconds. We used the software approach for transmission error rate measurement by sending 2000 short packets between the two nodes. The time used for sending "measurement" packets (see Section3.2) alone took 0.45-0.5 seconds. When we used fewer short packets for measurement, the time spent on this procedure can be cut down proportionally. If the hardware BER measurement scheme available on our experimental platform is used, the time used for measuring the link quality can be further shrunk. Timer setups may also be adjusted to reduce the time spent by the handshake protocol. However, the time to conduct entire mode switching processing is at least 200 ms based on our experiments, because the wireless packet transmission takes time. This time can be used to find the maximum allowable incoming traffic data rate under a given buffer size, when data loss must be avoided completely. Conversely, the time can be used to determine the size of the buffer given the data rate demanded by incoming traffic.

6. CONCLUSIONS

We built an FPGA-based wireless OFDM communication system that can switch its operation between different communication modes. The design is modular in that it can incorporate different wireless condition measurement schemes as a feedback to the system, and can use different mode switching techniques to transition between different modes. Tests in a lab environment showed that the design was robust, and satisfied the functional requirements. The system can be improved by further study on individual modules. For example, we can develop different schemes to measure various channel quality metrics, as may be demanded by distinct working environments and diverse system design goals. We may also explore different mode switching technologies and embed dynamic reconfiguration schemes, such as DPR, into our design.

7. ACKNOWLEDGMENTS

We are grateful to Dr. Jon Sjogren from Air Force Office of Scientific Research (AFOSR/NE) for his support on this project.

8. REFERENCES

[1] LAN/MAN Standards Committee of the IEEE Computer Society, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications High Speed Physical Layer in the 5GHz Band," *ANSI/IEEE Std 802.11*, 1999.

[2] C. Dick and F. Harris, "Fpga implementation of an ofdm phy," *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 1, pp. 905–909, Nov. 2003.

[3] WARP Team, Rice University, "http://warp.rice.edu."

[4] J. Terry and J. Heiskala, *OFDM Wireless LANs: A Theoretical and Practical Guide*, 2nd ed. Sams Publishing, July 2001.

[5] G. L. Stüber, J. R. Barry, S. W. McLaughlin, Y. G. Li, M. A. Ingram, and T. G. Pratt, "Broadband mimo-ofdm wireless communications," in *Proceedings of the IEE*, vol. 92, no. 2, Feb 2004, pp. 271–294.

[6] Lattice Semiconductor, "Implementation of an ofdm wireless transceiver using ip cores on an fpga," 2005.

[7] K. Masselos, A. Pelkonen, M. Cupak, and S. Blionas, "Realization of wireless multimedia communication systems on reconfigurable platforms," *J. Syst. Archit.*, vol. 49, no. 4-6, pp. 155–175, 2003.

[8] J. Park, H. Jung, and V. Prasanna, "Efficient fpga-based implementations of mimo-ofda physical layer," in *ERSA*, 2006.

[9] S. Koh and O. Diessel, "The effectiveness of configuration merging in point-to-point networks for module-based fpga reconfiguration," in *Proceedings of FCCM*, 2008, pp. 49–60.

[10] W. Fu and K. Compton, "Scheduling intervals for reconfigurable computing," in *Proceedings of FCCM*, 2008, pp. 49–60.

[11] M. French, E. Anderson, and D.-I. Kang, "Autonomous system on a chip adaptation through partial runtime reconfiguration," in *Proceedings of FCCM*, 2008, pp. 49–60.

[12] Altera Corporation, "http://www.altera.com."

[13] Xilinx Incorporated, "http://www.xilinx.com."

[14] the VideoLAN team, "http://www.videolan.org."

[15] NLANR/DAST, "http://dast.nlanr.net/Projects/Iperf."