WARPLab: Multi-node Prototyping with Real Wireless Data[☆]

Melissa Duarte, Patrick Murphy, Christopher Hunter, Siddharth Gupta, and Ashutosh Sabharwal[1]

Abstract

In this paper we present WARPLab, a framework which allows rapid prototyping of algorithms for wireless communications by combining the ease of MATLAB with the capabilities of the Wireless Open-Acces Research Platform (WARP). The WARPLab framework provides the software necessary for easy interaction with the WARP hardware directly from the MATLAB workspace. WARPLab is unique in its capabilities for being used as both a General Purpose Processor (GPP)-Software Defined Radio (SDR) platform and a hybrid GPP-SDR and Programable Hardware (PH)-SDR platform. With WARPLab the user can choose to implement all of the baseband signal processing in MATLAB, hence, enabling the use of WARP as a GPP-SDR. The user can also choose to implement part of the signal processing in real-time on the WARP hardware's FPGA, hence, enabling the use of WARP as a hybrid GPP-PH-SDR platform. This paper demonstrates the capabilities of WARPLab by presenting examples of wireless communications algorithms

[☆]This work was partially supported by NSF grants CNS-0619769, CNS-0551692, and CNS-0923479

[1]M. Duarte, P. Murphy, C. Hunter, S. Gupta, and A. Sabharwal are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX, 77005 USA, e-mail: {mduarte, murphpo, chunter, sgupta, ashu}@rice.edu

implemented using WARPLab and a detailed description of the WARPLab architecture.

Keywords:
Software Defined Radio, Wireless Communications, FPGA, WARP.

---

1. Introduction

Next generation wireless communication systems are being designed to achieve higher data rates and reliability by using novel technologies like multiple antenna schemes [1], cooperative communication [2], and cognitive radio [3]. Deployment of these technologies requires prototyping and testing for proof-of-concept. Both industry and academia are adopting Software Defined Radio (SDR) platforms for evaluation of novel wireless technologies because of SDRs flexibility and programmability [4]. Two types of commonly used SDRs are General Purpose Processor(GPP)-SDR and Programmable Hardware (PH)-SDR platforms [5]. PH-SDRs are capable of implementing high speed DSP-intensive operations, whereas performance of GPP-SDRs is more limited since it depends on processing power of the GPP being used and the speed and latency at which samples are transferred from Analog to Digital Converters (ADCs) to the GPP. However, GPP-SDRs offer the benefit of high level programming, while PH-SDRs require more specialized hardware programming.

GNU Radio [6] and SORA [7] are examples of GPP-SDR platforms. WARP [8] is an example of a PH-SDR platform, however, due to its flexibility and extensibility, WARP can be configured as a GPP-SDR. In this paper we present WARPLab, a framework developed for WARP which en-

ables the use of WARP as a GPP-SDR by allowing interaction with WARP hardware directly from the MATLAB workspace. Using WARPLab, all the baseband signal processing can be done offline in MATLAB and transmission and reception of RF signals can be done in real-time over-the-air using WARP hardware. Furthermore, WARPLab gives the user the option of implementing processing blocks in real-time on the WARP hardware's FPGA. Hence, WARPLab allows use of WARP as a hybrid GPP-PH-SDR platform which allows FPGA implementation of time sensitive blocks and facilitates a piecewise MATLAB-code-to-hardware flow where prototype baseband processing can first be implemented in MATLAB code and then parts or all of the system can be moved to an FPGA implementation.

To demonstrate the capabilities of WARPLab, we will present a detailed description of the WARPLab architecture and three examples of our current use of WARPLab for prototyping of multiple antenna algorithms and cooperative communication. The first example shows the use of WARPLab as a GPP-SDR platform for implementation of an Alamouti system [9]. The second and third examples demonstrate the use of WARPLab as a hybrid GPP-PH-SDR platform for implementation of a feedback based beamforming system and a cooperative communication system.

2. WARPLab Architecture

The basic WARPLab setup is shown in Figure 1 wherein two WARP nodes and a host PC running MATLAB are connected to an Ethernet switch. One host PC can control multiple nodes, as shown in Figure 2. The basic WARPLab flow is as follows. The digital baseband samples to be transmitted

over-the-air are created by the user in MATLAB. The user downloads these samples from the MATLAB workspace to transmit buffers in the WARP nodes. The user sends, from the MATLAB workspace, a trigger to transmitter and receiver nodes. Upon reception of this trigger, samples stored in transmit buffers are transmitted over-the-air and captured in real-time. Transmitter radios perform real-time digital to analog conversion of baseband samples and real-time upconversion from analog baseband to analog RF. Receiver radios perform real-time downconversion from analog RF to analog baseband and real-time conversion from analog baseband to digital baseband. The captured (received) digital baseband samples are stored in receive buffers in the WARP nodes. The user reads captured baseband samples from the receive buffers in the WARP nodes to the MATLAB workspace and then baseband samples in the MATLAB workspace are processed offline.

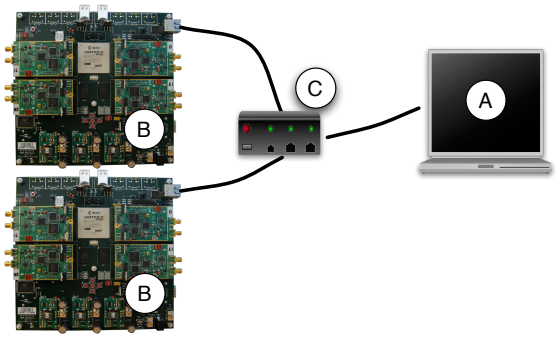

Figure 1: Basic WARPLab setup. A: Host PC, B: WARP node, C:Switch. The switch is connected to the WARP nodes and the host PC via Ethernet links.

The WARPLab framework provides the following code to the user.

- WARPLab Reference Design: contains all the code (FPGA code) required to program the WARP nodes.

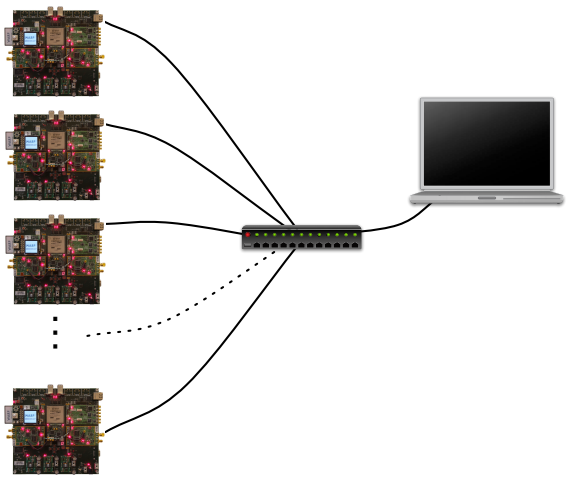

Figure 2: Multiple nodes can be controlled from one host PC.

- WARPLab Reference M-Code: contains MATLAB Code (M-Code) functions that allow interaction with WARP nodes from a host PC running MATLAB.

- WARPLab M-Code Examples: illustrate how to use the functions in the WARPLab Reference M-Code.

The WARPLab architecture is shown in Figure 3. The rest of this section explains the different parts of the WARPLab architecture and the code provided in the WARPLab Reference Design, Reference M-Code and M-Code examples.

2.1. WARP Hardware

The hardware for a WARP node with four radios is shown in Figure 4. The hardware consists of a WARP FPGA board and four radio boards. The WARP FPGA board and the radio boards were both designed at Rice

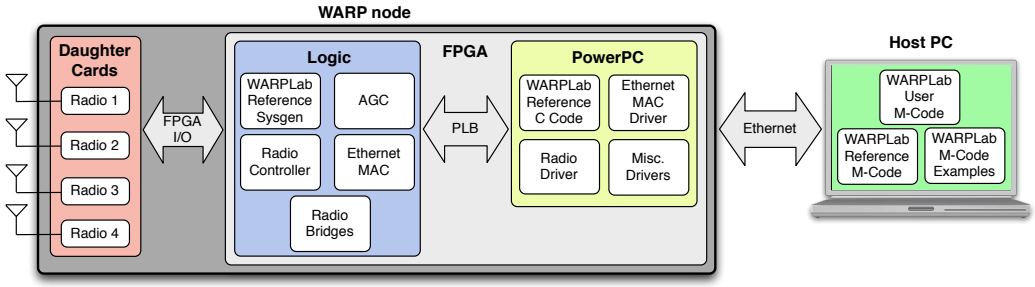

Figure 3: WARPLab Architecture.

University. Two versions of the WARP FPGA board have been developed, the first version (WARP FPGA board v1) is built around a Xilinx Virtex-II Pro FPGA and the second version (WARP FPGA board v2) is built around a more powerful Xilinx Virtex-4 FPGA. Figure 4 shows the WARP FPGA board v1; the WARP FPGA board v2 is a newer version released in 2009. Table 1 compares the FPGA and board resources respectively. This comparison and further documentation for the WARP FPGA boards can be found in [10].

Both versions of the WARP FPGA board have four daughter card slots. Each slot is connected to a dedicated bank of I/O pins on the FPGA. The four daughter card slots on the WARP FPGA board can be used to connect the FPGA to radio boards so that a four antenna node can be easily implemented. Daughter card slots are compatible across both versions of the WARP FPGA board. The radio boards will be described with more detail in Section 2.6.

The WARPLab architecture is the same for both versions of the WARP FPGA board. However, the WARPLab framework can achieve better performance when the WARP FPGA board v2 is used. This is because v2 of the FPGA board has a faster Ethernet link (enabled by the gigabit Ethernet

Table 1: FPGA and Board Resources

Parameter	FPGA Board v2	FPGA Board v1
FPGA Device	Virtex-4 FX100	Virtex-II Pro P70
FPGA Logic Slices	42k	33k
FPGA 18x18 Multipliers	160	328
FPGA 18kb Block RAMs	376	328
FPGA PowerPC Cores	2	2
FPGA Hard Ethernet MAC Cores	4	0
Daughtercard Slots	4	4
Memory	Up to 2GB (DDR2 SO-DIMM Slot)	Two 2MB ZBT SRAM
Ethernet	One Gigabit (10/100/1000) interface Two Gigabit (1000Base-T only) interfaces	One 10/100 interface

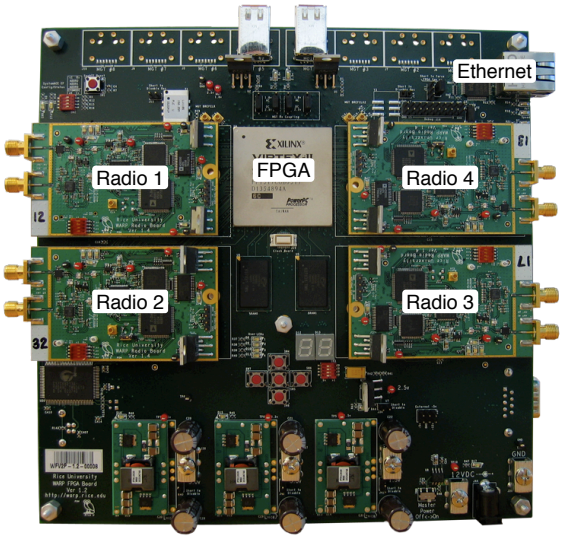

Figure 4: WARP node with four radios.

and hard Ethernet MAC cores) and a more powerful FPGA.

2.2. MATLAB Code

The MATLAB-Code (M-Code) consists of the WARPLab Reference M-Code, WARPLab M-Code Examples and the user M-Code. The WARPLab Reference M-Code is a set of M-Code functions that facilitate users' interaction with WARP nodes from the MATLAB workspace. As an example, consider the 'warplab_initialize.m' and 'warplab_writeSMWO.m' functions which are both part of the WARPLab Reference M-Code. The 'warplab_initialize.m' function establishes a UDP connection with WARP nodes that have an Ethernet link to the host PC, and UDP socket handles for WARP nodes become available from the MATLAB workspace. These UDP handles, together with other functions in the reference M-Code, simplify tasks like writing vectors of samples from the MATLAB workspace to buffers in the WARP nodes. For ex-

ample, calling the function 'warplab_writeSMWO(Node_handle, TxBuffer_ID, M_vector)' will write in the transmit buffer with ID 'TxBuffer_ID' in WARP node with handle 'Node_handle' the samples stored in the MATLAB vector 'M_vector'. The WARPLab M-Code Examples illustrate how to use the functions in the WARPLab Reference M-Code.

The user M-Code is the code the user writes in order to generate the digital baseband samples to be transmitted. Using the functions in the WARPLab reference M-Code the user can control, directly from MATLAB, the transmission and reception of these samples in real-time RF over-the-air using WARP nodes.

With WARPLab the user has the flexibility of transmitting narrowband signals or wideband signals using different modulation types and coding techniques. This flexibility is possible because the only requirements the digital baseband signal (samples) created by the user M-Code must satisfy are the following. 1) The digital baseband signal cannot have frequency components between -30 KHz and 30 KHz. This requirement is due to the frequency requirement of the MAXIM radio chip [11] used for RF upconversion and downconversion. 2) The digital baseband signal cannot have frequency components below -20 MHz or above 20 MHz. This requirement is due to the sampling frequency of the Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) which is fixed to 40 MHz. 3) The amplitude of the real part (in-phase component) of the digital baseband signal must be between -1 and 1 and the amplitude of the imaginary part (quadrature component) of the digital baseband signal must be between -1 and 1. This requirement is due to the input data type requirement of DACs [12].

WARPLab allows transmission of any digital baseband samples that satisfy the three requirements previously mentioned.

2.3. Ethernet

An Ethernet link is used for communication between the host PC and the WARP nodes. Peter Rydesater's open-source TCP/UDP/IP toolbox for MATLAB [13] is used to set UDP connection with WARP nodes and send and receive UDP packets from the MATLAB session running in the host PC. As stated earlier, Ethernet communication is faster when using the WARP FPGA board v2, which allows a gigabit Ethernet connection.

2.4. FPGA PowerPC Code

The code required to program the FPGA PowerPC (PPC) is provided as part of the WARPLab Reference Design. The PPC contains the WARPLab Reference C-code and drivers code, as shown in Figure 3. The WARPLab Reference C-Code implements an infinite loop that is listening for Ethernet packets. When an Ethernet packet is received, the ID (or opcode) of the command/action to be executed is read from the received packet. Based on this ID, the C code enters a specific case statement. When the case statement is executed the code returns to its initial state of listening to Ethernet packets.

To illustrate with more detail how the WARPLab Reference C code works, consider again the case in which the data stored in the MATLAB vector 'M_vector' is to be downloaded to a buffer in a WARP node. As previously mentioned, this can be done by calling the function 'warplab_writeSMWO(Node_handle, TxBuffer_ID, M_vector)' from the MATLAB workspace. This function call sends an Ethernet packet that contains the 'Node_handle',

'TxBuffer_ID', and 'M_vector' information. All the WARP nodes connected to the host PC ignore this Ethernet packet except the one that corresponds to 'Node_handle'. This node then parses the received packet and extracts the 'TxBuffer_ID' field, this is the ID (or opcode) field which specifies that the action to be executed corresponds to writing to the 'TxBuffer_ID' buffer. Hence, the C code enters a case statement that runs the lines of code required to execute this action. Code Example 1 shows the pseudocode for the WARPLab Reference C Code and for the case statement that writes the 'TxBuffer_ID' buffer.

Code Example 1: Writing the TxBuffer_ID

```
while 1 do
    Listen for Ethernet packets;
    if Received a valid packet then
        switch Command do
            case Command 1
                Statements to execute Command 1;
            case Command 2
                Statements to execute Command 2;
            case TxBuffer_ID
                Copy 'M_vector' received in Ethernet packet to transmit buffer with ID equal
                to 'TxBuffer_ID';
                Send acknowledgment to host PC;
            ...
            case Command N
                Statements to execute Command N;
```

2.5. FPGA Logic

The FPGA logic consists of hardware on-chip peripherals, e.g. Automatic Gain Control (AGC), and logic to interface off-chip hardware peripherals, e.g. radio daughter cards and the Ethernet connection. All hardware peripherals are connected to the Processor Local Bus (PLB). Through memory reads and writes, the software running on the FPGA PPC can pass values to peripherals and vice versa.

The WARPLab Reference Sysgen Core is an on-chip peripheral that contains the transmit and receive buffers that store baseband samples. This core is built using Xilinx's System Generator (Sysgen) tool. Transmitter and receiver buffers in the WARPLab Sysgen Core are connected to the radio boards, as shown in Figure 5. Baseband samples stored in the transmit buffers are read to the radio DACs in real-time. Similarly, baseband samples from the radio ADCs are written to the receive buffers in real-time. Read/write of transmit/receive buffers is triggered by commands sent by the user from the MATLAB session running in the host PC.

2.6. Radios

The radios are capable of targeting both the 2.4 GHz and 5 GHz ISM bands. The radios are designed for Multiple-Input Multiple Output (MIMO) antenna applications, guaranteeing the phase coherency of carriers generated by chips which share a reference clock. Figure 5 shows a block diagram for the transmitter and receiver paths for a radio board. The carrier frequency for upconversion and downconversion and the gain of the variable gain amplifiers can be set by the user from the MATLAB workspace using functions provided in the WARPLab Reference M-Code.

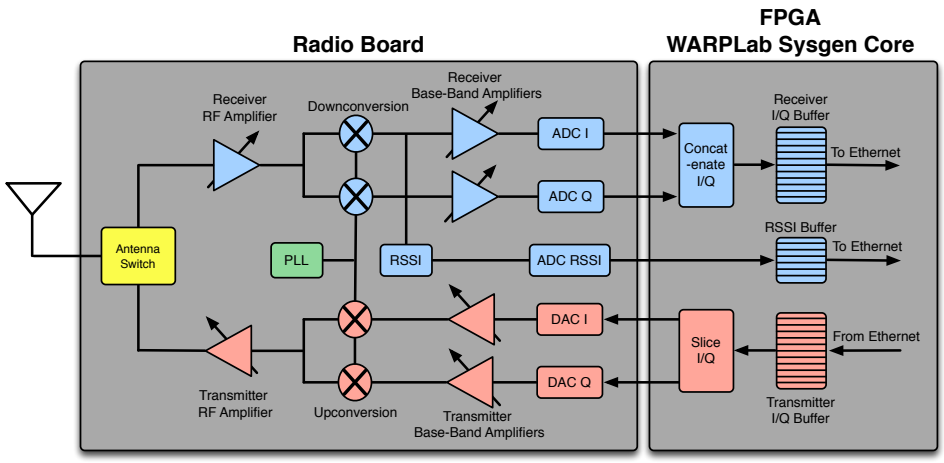

Figure 5: Transmitter and receiver paths for one radio.

2.7. Extending WARPLab

The WARPLab Reference Design, the WARPLab Rerence M-Code, and the WARPLab M-Code examples are all available in the WARP repository [8]. Users are encouraged to modify the WARPLab Reference Design and WARPLab Reference M-Code in order to meet the requirements of their experiment. For example, if part of the signal processing requires real-time implementation, then this processing can be implemented in FPGA logic by modifying the WARPLab Reference Design. The WARP repository provides tutorials and documentation to help familiarize with the tools required to program the FPGA.

3. WARPLab Examples

In this section we present examples that highlight the two main uses of WARPLab: WARPLab as a GPP-SDR platform and WARPLab as a hybrid GPP-PH-SDR Platform.

3.1. Using WARPLab as a GPP-SDR Platform

We have used WARPLab to implement a narrowband Alamouti system with two transmitter antennas and one receiver antenna (2×1 system). Alamouti [9] is a well known multiple antenna scheme that improves reliability in wireless systems by exploiting transmitter diversity. For the narrowband Alamouti implementation we used the WARP FPGA board v1. All the baseband processing was done in the MATLAB session running on the host PC, hence, for this experiment, the WARPLab framework was exercised as a GPP-SDR Platform.

The experiment setup is shown in Figure 6. The basic WARPLab setup is connected to an Azimuth channel emulator [14, 15]. Two radios at the transmitter node are connected to the channel emulator which combines the two RF signals to emulate a 2×1 wireless system. The RF output of the channel emulator is connected to a radio at the receiver node. Two RF channels are emulated, one from transmitter radio one to the single receive antenna (Tx1-Rx1 channel), and another channel from the second transmitter radio to the receiver (Tx2-Rx1 channel). The two RF channels are modeled as independent channels. The received signal is the superposition of the signal that travels over the Tx1-Rx1 channel and the signal that travels over the Tx2-Rx1 channel. This is the same channel model considered in [9]. During the experiment, the emulator output power was controlled from the host PC running MATLAB. More details on the experiment conditions and experiment results are presented and analyzed in [16]

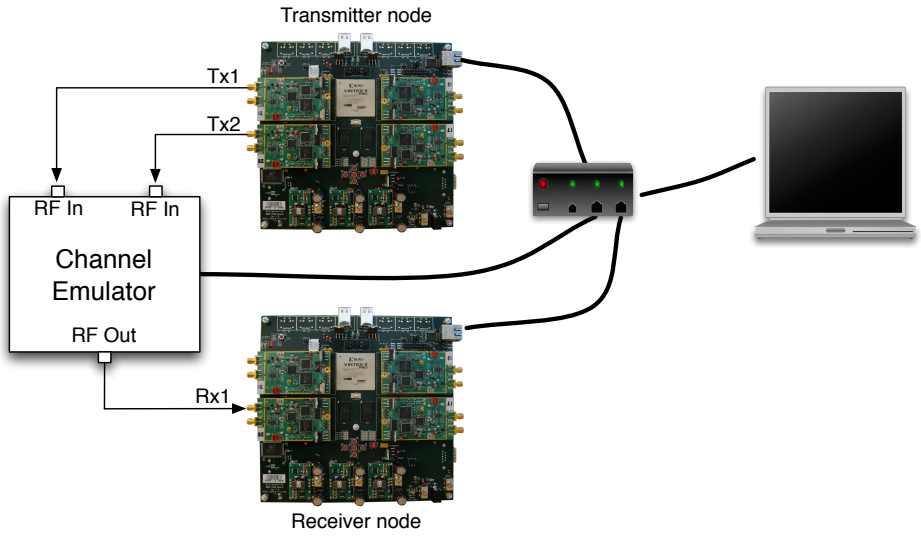

Figure 6: Experiment Setup.

3.2. Using WARPLab as a hybrid GPP-PH-SDR Platform

Allowing the user to implement part of the signal processing in MATLAB and part of the signal processing in the FPGA is one of the main features of WARPLab. In this section we present two examples that take advantage of this feature.

3.2.1. Beamforming Experiment

We implemented a narrowband 2×1 codebook based beamforming system as described in [17]. This system takes advantage of transmitter diversity and it is known to have better performance than the 2×1 Alamouti system. Transmission of signal x over two transmitter antennas using beamforming vector $[b_1, b_2]$ is implemented by transmitting $b_1 x$ over antenna 1 and $b_2 x$ over antenna 2. In a codebook based beamforming system, the transmitter sends a training signal to the receiver, the receiver uses the received training

signal to estimate the channels. Based on the channel estimates, the receiver chooses which beamforming vector to use in order to improve the Signal to Noise Ratio (SNR). A codebook, known at the transmitter and the receiver, contains all the possible beamforming vectors the receiver can choose. The receiver runs an exhaustive search over the codewords (or beamforming vectors) in the codebook to find the one that results in the best SNR for the given channel conditions. The index of the beamforming vector chosen by the receiver is fed back to the transmitter. Finally, the payload signal is transmitted using the beamforming vector corresponding to the index fed back from the receiver.

The feedback delay is the time between transmission of the training signal and transmission of beamformed payload. It has been shown that feedback delay affects the performance of a beamforming system [18]. Implementing a beamforming system in a PH-SDR platform will result in less feedback delay compared to a GPP-SDR platform implementation. However, PH-SDR implementation requires specialized hardware programming. Using WARPLab it is possible to implement a beamforming system using a hybrid GPP-PH-SDR implementation where most of the signal processing is implemented in MATLAB and the feedback delay can be reduced by implementing part of the signal processing in real-time in the FPGA. Below we describe how a simple change in the WARPLab FPGA Code (WARPLab Reference Design) allowed us to significantly reduce the feedback delay for the narrowband beamforming experiments.

Let $\mathbf{x}$ denote the vector of baseband payload samples to be transmitted and $[b_1, b_2]$ the beamforming vector to use for transmission. If all the

baseband processing happens in MATLAB, then vectors $b_1\mathbf{x}$ and $b_2\mathbf{x}$ are computed in MATLAB and then downloaded to the transmit buffers. For our experiments, vector $\mathbf{x}$ consists of 2^{14} samples and downloading the beamformed samples $b_1\mathbf{x}$ and $b_2\mathbf{x}$ requires a total of 1.4 s. This results in a very large feedback delay of a total of 1.45 s. Clearly, the main cause of the large feedback delay is the time it takes to download the beamformed vectors to the transmit buffers.

To reduce the feedback delay we do a very simple change on the WARPLab FPGA code: add two complex multipliers. Specifically, the path between transmitter buffer and radios shown in Figure 5 is modified as shown in Figure 7. Since only two radios are used, we only have to add two complex multipliers. Hence, we only have to download two complex values, namely b_1 and b_2, and computation of $b_1\mathbf{x}$ and $b_2\mathbf{x}$ happens in real-time as samples are read from transmit buffers to radio DACs. The time it takes to download b_1 and b_2 from the host PC to the transmitter node is significantly less than the time required to download the beamformed payload vectors $b_1\mathbf{x}$ and $b_2\mathbf{x}$ each consisting of 2^{14} complex values. Consequently, adding the two complex multipliers allows us to reduce the feedback delay from 1.45 s to 75 ms. We note that vector $\mathbf{x}$ is downloaded to the tansmit buffers before transmission of training, hence, the time required to download vector $\mathbf{x}$ doesn't affect the feedback delay.

In summary, simply moving the computation of the beamformed payload vectors from MATLAB to real-time time allows a significant reduction in feedback delay. The rest of the baseband signal processing (channel estimation, search for beamforming vector, modulation, demodulation, etc.) is done

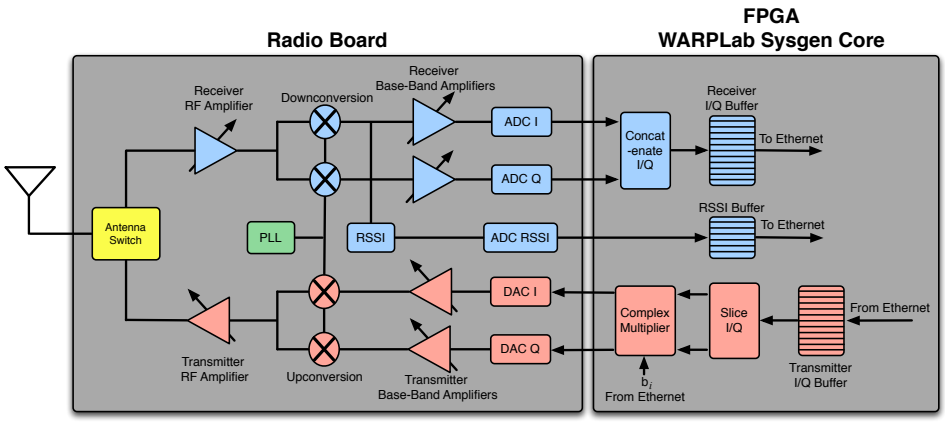

Figure 7: Modified WARPLab Reference Sysgen Core. The modification consists in adding a complex multiplier to the transmitter path.

in MATLAB. We note that for the beamforming experiments we used the WARP FPGA board v1, which doesn't have a fast Ethernet interface. We expect that feedback delay can be further reduced when implementing on the WARP FPGA board v2 since it includes faster Gigabit Ethernet and Hard Ethernet MAC. The results of the beamforming experiments are presented and analyzed in [16].

3.2.2. Cooperative Communications Experiments

One of our ongoing projects is the investigation of applying the techniques of cooperation in real communications systems. Cooperative communications is an approach to improving the performance of a wireless system by pooling the resources of multiple nodes. By coordinating their receptions and transmissions, cooperating nodes can realize performance gains similar to those enabled by non-cooperative nodes equipped with multiple-antennas. Our ultimate goal was the construction of a real-time cooperative Orthogonal

Frequency Division Multiplexing (OFDM) system, built around an FPGA implementation of a full physical layer transceiver. But before attempting to design the real-time transceiver, we needed to verify a few key assumptions about the behavior of cooperating nodes, a task for which WARPLab was ideally suited. We designed two WARPLab experiments in this effort.

Cooperative communications is often described as a distributed version of standard MIMO communication techniques. One area where this analogy breaks down is the problem of carrier frequency offset (CFO). In a true MIMO system, multiple antennas on a transmitter share the same clock that generates their carriers. In cooperative applications, devices may be topologically separated and do not have the luxury of sharing this common clock source. In our first experiment, we used WARPLab to show that the CFO problem can be solved passively in amplify-and-forward instead of decode-and-forward relays [19]. We used two nodes, each running the standard WARPLab Reference Design. The first node transmitted a constant baseband waveform, resulting in an RF waveform consisting of just the carrier sinusoid itself. The second node received this transmission, captured the I/Q samples, then re-transmitted the same samples. The first node received this re-transmission and offloaded the captured waveform to MATLAB. As expected, the waveform received at the first node was a constant baseband signal, with no frequency offset imposed by the receive-transmit operations at the second node.

Our second experiment sought to verify that an amplify and forward relay could capture and re-transmit a wideband OFDM waveform of sufficient quality to actually improve the reliability of a wireless link. This experiment

utilized three WARP nodes. Two, those acting as the source and destination nodes, used the standard MATLAB-controlled WARPLab reference design. The third node, acting as the relay, used a custom WARPLab design which implemented automatic, sequential receive and transmit phases. The receive phase was triggered by MATLAB and coincided with the first half of the source node's transmission. In this phase, the relay captured the raw received I/Q samples. The relay's transmit phase followed immediately thereafter, coinciding with the second half of the source's transmission. In its transmit phase, the relay re-transmitted the raw samples previously captured. The destination node captured both transmissions (from the source in slot one, and from the source and relay in slot two). The captured samples were offloaded to MATLAB for processing. This scheme mimicked the two time slot schedule of a real-time amplify and forward cooperative system while preserving the flexibility of MATLAB for all physical layer processing. Using this setup, we were able to verify that, over a useful range of SNRs, the relay's re-transmission reproduced the original waveform with sufficient quality to enable successful decoding at the destination. This was a key result, providing confidence that a comparable real-time implementation could realize actual performance gains in a real network of cooperative nodes.

The cooperative communication experiments described above took advantage of the use of WARPLab as a framework that enables WARP as a GPP-SDR platform. In these experiments, using MATLAB for all physical layer processing and using WARP hardware for over-the-air transmission and reception facilitated the implementation of our first prototype of cooperative communication systems. However, reading and writing samples between the

MATLAB workspace and the WARP FPGA buffers resulted in a minimum receive-to-transmit delay of approximately 1 second. This delay was acceptable for a first prototype but our final goal was the implementation of a real-time system. Hence, we ported the physical layer processing from MATLAB code to FPGA code. Using WARPLab as a hybrid GPP-PH-SDR platform facilitated porting the physical layer processing from MATLAB code to FPGA code because both implementations use exactly the same WARP hardware. We recently completed the design of a cooperative communication system in which the physical layer runs in real-time in the WARP FPGA [20]. Furthermore, we have used this physical layer in the study of cooperative medium access control [21, 22]. In these designs the time between reception and transmission is approximately 20 μs. Given that our design operates in a 10 MHz bandwidth, a receive-to-transmit delay of 20 μs is comparable to the delay in Wi-Fi systems.

3.3. More WARPLab Examples

We have presented examples that illustrate the use of the WARPLab framework. WARPLab is currently being used by several research labs. Authors in [23] have proposed a Cooperative Partial Detection (CPD) strategy for detection in multiple antenna relay channels and verified the performance of CPD using WARPLab. A Cooperative Physical layer protocol has been demonstrated in [24]. More research results using WARP and WARPLab can be found at [25].

4. Conclusion

We have created a framework for rapid prototyping of algorithms for wireless communications. The framework gives the user flexibility to implement baseband signal processing on a GPP or in real-time on an FPGA. The framework is currently being used by several research groups.

References

[1] Zheng, L., Tse, D.. Diversity and multiplexing: A fundamental tradeoff in multiple-antenna channels. IEEE Transanctions on Information Theory 2003;49:1073–1096.

[2] Kramer, G., Maric, I., Yates, R.. Cooperative communications. Foundations and Trends in Networking 2006;1(3).

[3] Haykin, S.. Cognitive radio: brain-empowered wireless communications. IEEE JSAC 2005;23(2):201–220.

[4] Schmid, T., Sekkat, O., Srivastava., M.B.. An experimental study of network performance impact of increased latency in software defined radios. In: WinTECH '07: Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization. 2007, p. 59–66.

[5] Miljanic, Z., Seskar, I., Le, K., Raychaudhuri, D.. The WINLAB network centric cognitive radio hardware platform - WiNC2R. In: 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications, CrownCom. 2007, p. 155–160.

[6] GNU Radio. 2010. URL http://gnuradio.org.

[7] Tan, K., Zhang, J., Fang, J., Liu, H., Ye, Y., Wang, S., et al. SORA: high performance software radio using general purpose multi-core processors. In: NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation. Apr.; 2009, p. 75–90.

[8] WARP repository. 2010. URL http://warp.rice.edu/trac.

[9] Alamouti, S.M.. A simple transmit diversity scheme for wireless communications. IEEE JSAC 1998;16(8):1451–1458.

[10] Mango Communications. 2010. URL http://www.mangocomm.com/products/boards/warp-fpga-board-v2.

[11] Maxim MAX2829. 2011. URL http://www.maxim-ic.com/max2829.

[12] WARPLab framework. 2010. URL http://warp.rice.edu/WARPLab.

[13] 2010. URL http://www.mathworks.com/matlabcentral/fileexchange/345.

[14] Azimuth ACE 400WB. 2010. URL http://www.azimuthsystems.com/platforms-channel-400wb.htm.

[15] Celine, G.. Effectively testing MIMO-enabled wireless devices. In: RF DESIGN; vol. 30. 2007, p. 40–44.

[16] Duarte, M., Sabharwal, A., Dick, C., Rao, R.. Beamforming in MISO systems: Empirical results and EVM-based analysis. IEEE Transactions on Wireless Communications 2010;9(10):3214–3225.

[17] Love, D.J., Heath, R.W., Strohmer, T.. Grassmannian beamforming for multiple-input multiple-output wireless systems. IEEE Transanctions on Information Theory 2003;49:2735–2747.

[18] Ma, Y., Zhang, D., Leith, A., Wang, Z.. Error performance of transmit beamforming with delayed and limited feedback. IEEE Transactions on Wireless Communications 2009;8(3):1164–1170.

[19] Murphy, P., Sabharwal, A., Aazhang, B.. On building a cooperative communication system: Testbed implementation and first results. EURASIP Journal on Wireless Communications and Networking 2009;.

[20] Murphy, P., Sabharwal, A.. Design, implementation and characterization of a cooperative communications system. IEEE Transactions on Vehicular Technology 2011;.

[21] Hunter, C., Murphy, P., Sabharwal, A.. Real-time testbed implementation of a distributed cooperative MAC and PHY. In: Proceedings of CISS. 2010,.

[22] Hunter, C., Sabharwal, A.. Distributed protocols for interference management in cooperative networks. IEEE Journal on Selected Areas in Communications, *to appear*, 2012;.

[23] Amiri, K., Wu, M., Duarte, M., Cavallaro, J.. Physical layer algorithm and hardware verification of MIMO relays using cooperative partial detection. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP. 2010,.

[24] Singh, S.R., Siddiqui, E.A., Korakis, T., Liu, P., Panwar, S.S.. A demonstration of video over a cooperative PHY layer protocol. In: ACM MobiCom. 2008,.

[25] WARP papers and presentations. 2010. URL `http://warp.rice.edu/papers`.