Software for a Full-Duplex Communication System*

Jayanth Shenoy and Dr. Sriram Vishwanath (supervisor)
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas , USA
jayanth.shenoy@utexas.edu

***Abstract* – Currently in development, the pre-5G communication system built by the Austin startup GenXComm involves the use of self-interference canceling (SIC) technology. Through this phenomenon, the SIC radio prototypes are able to cancel out the noise produced by each other and create a full-duplex communication effect. The system is more optimized in both speed and reliability due to reduced noise. However, there are a great deal of steps and challenges involved in the process of perfecting the development of the communication system. This paper primarily explores the software behind the application layer for the SIC radio system. The main tasks of the software involve the following: (1) data transfer between the computer application and embedded system over the ethernet, (2) data transfer over the air between embedded devices using the SIC principle. In order to fully illustrate these capabilities of the SIC communication system, a video streaming demo was built as the software application layer to run on the SIC radio system.**

Index Terms – full-duplex communications, self-interference cancellation, embedded system, pre-5G, packet streaming

I. Introduction

Never before have both the speed and reliability of our communication systems been more important. In a world full of data, the task of processing and transmitting information has become critical. As a result, more and more engineers are continuing to join the development of proper 5G communication systems. Because of the complexity of developing communication systems, engineers with a variety of different technical expertise are involved in the development.

A. Background on Full-Duplex Communications

A primary goal of 5G communication systems is to increase the speed of data transmitted and received through full-duplex communication. Most non-5G communication systems today are only half-duplex, essentially meaning that the radios on each end of the communication system can either send data or receive data but cannot perform both at the same time. The reason why half-duplex communication systems cannot perform the two tasks simultaneously is because noise is produced whenever a radio attempts to send data, thus preventing it from accurately receiving any incoming data. Consequently, data transfer rates in a half-duplex communication system are significantly slower.

On the other hand, a full duplex communication system has radios that are able to simultaneously perform the task of sending and receiving data. The primary means through which researchers are able to create the full-duplex effect is through self-interference cancellation (SIC). This theory is achieved when each of the radios attempts to generate a signal that cancels out the noise produced while talking. The end result is a much faster and more reliable communication system.

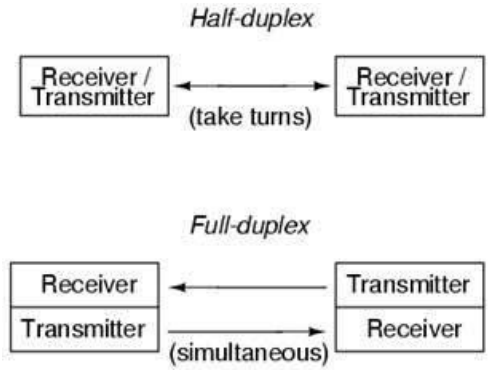


Fig. 1. Diagram of half-duplex and full-duplex systems [1]

B. The GenXComm SIC Radios

In recent years, researchers have made attempts to construct full-duplex communications systems. However, many issues regarding the design and structure of these radios have prevented them from becoming more mainstream. These issues include costs of specialty hardware, energy consumption of devices, and unreliability of systems in harsh environments.

Unlike most solutions to full-duplex communications, the SIC radios developed by GenXComm are able to achieve the full-duplex effect while solving the issues mentioned above. They consume significantly lower amounts of power while maximizing the use of the frequency spectrum. Additionally, the SIC radios lend themselves to being more scalable and compatible with existing technology.

Img. 1. GenXComm SIC Radio

* This work is supported by the NSF grant for the Extensible Undergraduate Research in Communications Applications REU

II. Background on SIC Communications System

A. *Data Path of the Communications System*

A basic system of the SIC radio network includes two computers and two embedded devices. On each of the computers, an application runs where the user is able to interface with certain settings of the communication system and transmit data to the other computer. Once the user's changes are made on the application, the new settings data and payload will be bundled by the application and sent to the radio's embedded device via ethernet.

Once the embedded device has received the packet, the settings data will be processed by the embedded system's internal hardware, and the rest of the packet will be transmitted to the second radio over the air via full-duplex communication. After the second radio receives the packet, the data will then be transmitted back up to the second computer via ethernet. The application on the computer will parse the packet and display its results.

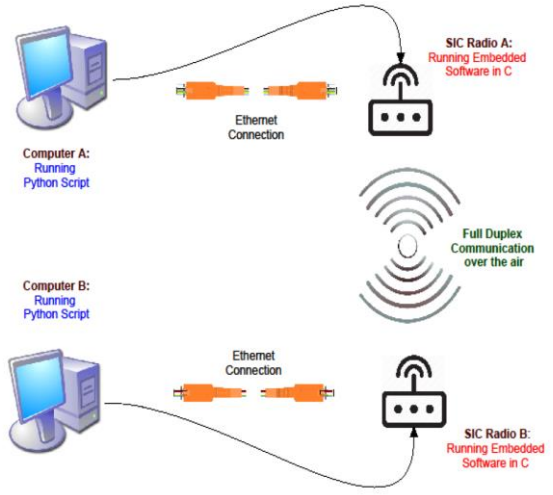


Fig. 2. System Diagram of SIC Radio network

B. *Hardware Choices, Specs and Implementation*

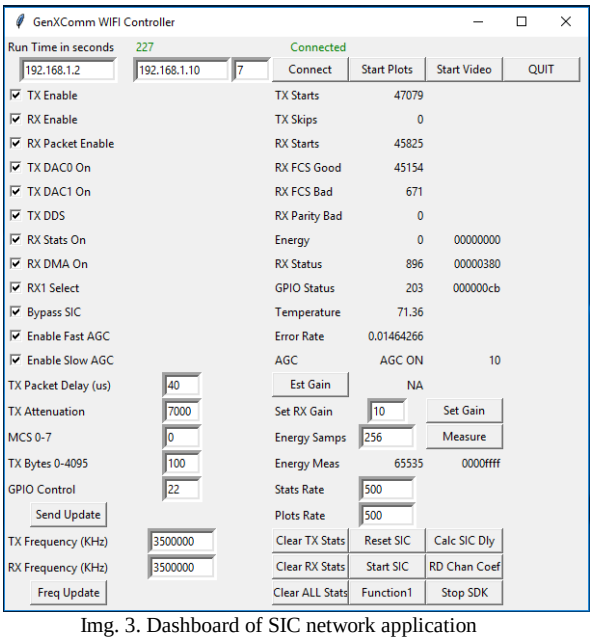
In addition to the ordinary desktop computers, the hardware for the communications network included two embedded devices. In order to simplify the prototyping processes, these hardware devices were primarily off-the-shelf models.

The first device that was used was the Xilinx Zedboard. Though it is the cheapest embedded device we used, we quickly realized that it was not sophisticated or powerful enough to handle the SIC communications network. Eventually, we moved onto the Xilinx ZC706 and the Xilinx Picozed. Both of these embedded devices have on-board Field Programmable Gate Arrays (FPGAs). Hardware definition files for the FPGAs were created in Vivado using Verilog. Before testing any software, the proper bitstream needed to be uploaded to the board. The radios from the demo in Img. 1 contain the Xilinx Picozed, which was less bulky and not as powerful as the Xilinx ZC706. Typically, the embedded device was connected to the computer with an ethernet cable and a J-tag to USB cable.

Img. 2. Xilinx Picozed [2]

C. *Software Choices, Specs and Implementation*

As stated above, the communications network consists of two primary software components: (1) the computer application and (2) the embedded software. For the computer application, we chose to use Python. It is a simple language where much can be accomplished with only a few lines of code. The graphics library that was primarily used was Tkinter. Many labels, buttons, and text-fields on the application dashboard were created using a grid layout. Features on the application dashboard include the abilities to change network connection settings and display network statistics. The application dashboard also has buttons to launch the video display demo. The video display demo was developed using a Python graphics library known as OpenCV, and the transport layer protocol for the video streaming was the User Datagram Protocal (UDP). A good portion of the application was debugged in the Python terminal, but some portions of the application were developed in PyCharm.



Img. 3. Dashboard of SIC network application

III. Process of Software Development

A. *Coding a Video Demo*

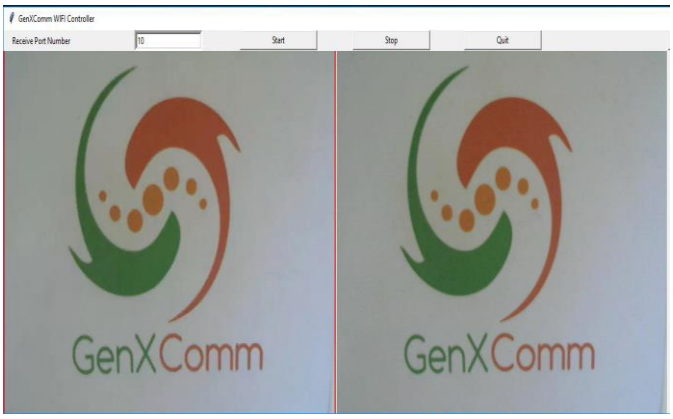
To best illustrate the features of the SIC radio network, a video demo needed to be created as part of the network's application layer. Much of the video demo's content in the Python application was implemented by using the library OpenCV. Essentially, a video is a series of images that are taken and streamed extremely quickly after each other. With OpenCV, the Python application was able to gather image data from the desktop camera. From there, a packet was constructed. Since the image frame from OpenCV contained more than 1500 bytes (the maximum size of a packet transmitted through ethernet), the frame needed to be parsed and sent in multiple packets.

When sending the video to the embedded system, the chosen transfer layer protocol to use was UDP. Since keeping a consistent and fast stream of images was more important than transmitting every exact image perfectly, this protocol was perfect to implement with a video demo.

In Python, network implementation was done with the help of sockets. Every time the application was tested on digital loopback, two instances of the application needed to be simultaneously running: (1) a client runner (2) a server runner. On the client side, images would be taken to the laptop and sent through the SIC network while the server side would receive, restore and display the segmented image frames.

Options on the video dashboard store below allowed the user to start, stop, and quit the video streaming application. As long as a valid camera was detected, the application would begin successfully, and the video dashboard would launch. If a valid camera was not detected, the OpenCV functions would throw exceptions.

The "backend" embedded code for the video demo in C was not too complex. Simply, a call back function was used in the main loop to fetch the UDP packet from the ethernet and place it in a TX buffer for over-the-air transmission. The Xilinx LWIP library was able to provide most of these functions and framework that would have been otherwise more time-consuming to write.



Img. 4. SIC Video application Transmit and Receive Window Side by Side

B. *Debugging Software*

One of the most complex issues in debugging the SIC network was determining whether the issue was hardware-related or software-related. Many times, we examined CV files and packet analyzing tools such as Wireshark to determine what was being sent and whether it was being received correctly. In several cases where there was a packet data mismatch, the hardware had issues as there were errors in the built hardware definition files for the project.

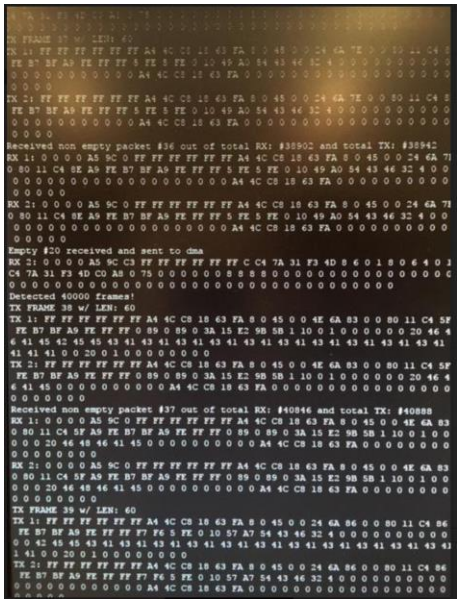


Fig. 3. CV file of packet transportation.
Data in the first few packets here are corrupted.

On the other hand, many logical bugs in the networking system were solely due to errors in the software. Oftentimes, these errors were the most complicated to both find and fix. To fix a software issue, the first issue that would need to be determined was whether the bug was in the code for the application or in the code for the embedded software. After that, finding the bug was essentially a matter of commenting out pieces of code to see which snippets resulted in the network error.

One of the main issues that we needed to fix, for example, occurred with the application dashboard. The Matplot graphs on the screen did not seem to be displaying the correct network statistics, despite the successful connection of the network and video demo. At first, we believed that there was a bug in the application code. It took us several days to sift through all of it and realize that nothing was wrong with the application code. Instead, we found that data being sent from the embedded device was storing itself in a wrong memory location. Contrary to our original beliefs, the simplest way to fix this bug was to change several lines in the embedded software so that the statistics packets were being stored in the right place for the application to read and display. In short, the complex debugging process of the network software was the most time-consuming aspect of the full-duplex Wi-Fi project.

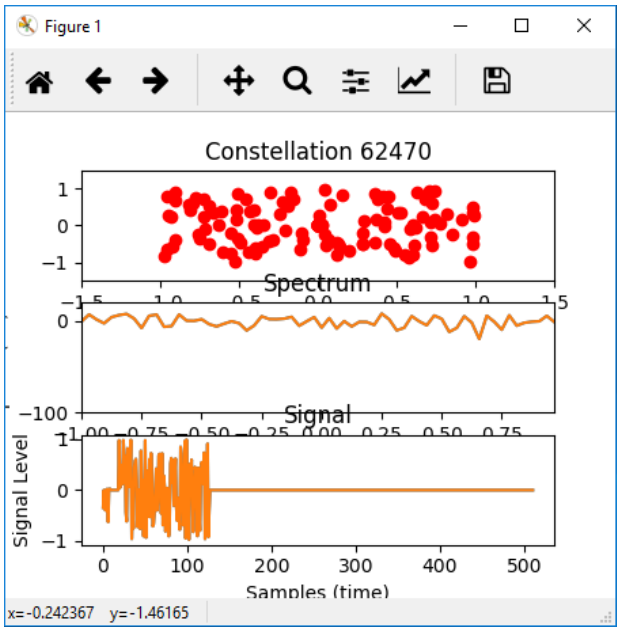


Fig.4. Matplot graph showing statistics information for the connection including signal strength and constellation data.

III. Future of SIC Radio Software

A. *Issues with The Current System*

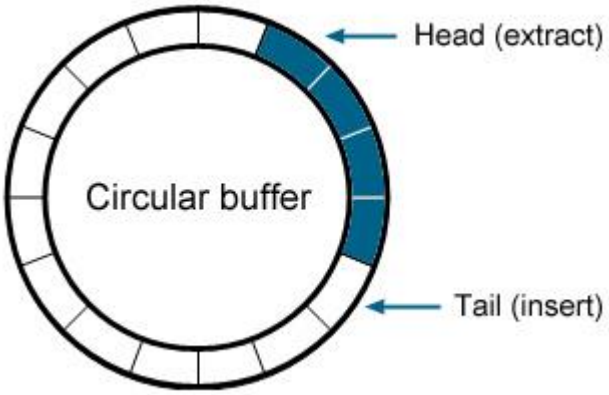
Although the current SIC radio software is successfully working, there are many issues with the current software that cannot be simply ignored. Several issues include:

1. Video demo is extremely slow in streaming images. Additionally, although correct images are being displayed, some of the frames are slightly "foggy" and pixelated due to packet loss.
2. The graphical user interface for the dashboard, although simple, looks quite dull and ordinary. Most graphical components were simply placed without much thought towards design or aesthetics.
3. Much of the embedded software uses a busy-wait to search for a flag sent by the application for hardware statistics data. Making this system interrupt-driven would be both more efficient and sophisticated.

B. *A New Software Design*

Most of the issues above exist because of flaws in the current software design rather than bugs in the code. As a result, we decided to redesign most of the network's software. Most of the connection issues and packet loss in the current software stems from using the LWIP to bundle the packets. For the future software, we decided to not use the LWIP all together. Instead, the software on the embedded system will manually send the packets over the air through frame-by-frame parsing. By manually parsing and sending the packets, we will be able to reduce the time delay between image frames of the video demo. The new, additional software files required for the front-end and backend of the new communication system are as follows:

1. cyclic_buffer.c – Provides code for a general use cyclic buffer, containing pointer information to memory locations of various stored packet frames.
2. frame_dma.c – Provides code for Direct Memory Access (DMA) controls. Many functions and operations described in this file relate to the movement of packets between ethernet, memory, and Wi-Fi through DMA.
3. xemacps.c – Provides code for the ethernet RX and TX blocks. Interrupt-based bits trigger functions in this file once packets are ready to be sent or received via ethernet.
4. gxc_sic_driver.c – The SIC driver controls all the processes that occur after the packet data is transferred through DMA. These processes include interrupt and DMA initializations, memory management, and maintenance of cyclic buffers.



Img.5. Concept of the cyclic buffer with two access pointers. The implementation complexity is much lower than that of a standard linear buffer. [3]

IV. CONCLUSION

The future of communications technology relies on the success of the full-duplex radio implementation. The GenXComm radios accomplish creating the full-duplex effect through the concept of Self-Interference Cancellation. Software for the SIC communications network involves both a front-end application component and a back-ended embedded component. The application fetches user inputs on a computer using a Python Tkinter GUI, bundles the data into packets and sends those packets to the embedded system via ethernet. The embedded software, written in C, then processes the received packets and then sends them to the other embedded device in the communication network via SIC Wi-Fi. To fully demonstrate the capabilities of the communication system, a video streaming software was built as the application layer. Although the quality of the video demo is poor in the current system, the future software layer for the radio reduces the

amount of overhead data added to each packet, allowing for a higher quality video demo and a more powerful communications system.

ACKNOWLEDGMENT

This research and development was made possible with the support of The University of Texas at Austin Department of Electrical and Computer Engineering, as well as through funding from the National Science Foundation (EURECA REU grant). Additionally, I would like to thank members of the GenXComm team including Dr. Sriram Vishwanath, Hardik Jain, and Ousek Son.

REFERENCES

[1] Editor, T. (2017, May 1). [Half-Duplex Full-Duplex]. Retrieved March 10, 2018, from http://www.polytechnichub.com/what-is-a-duplex-communication/

[2] PicoZed SDR [Digital image]. (2017). Retrieved March 10, 2018, from http://docplayer.net/50677864-Picozed-sdr-breakout-carrier-card-hardware-user-guide-version-1-2.html

[3] [Circular Buffer]. (n.d.). Retrieved March 10, 18, from http://www.camera-sdk.com/p_54-how-to-implement-circular-buffer-video-recording-in-c-onvif.html