CRUSH: COGNITIVE RADIO UNIVERSAL SOFTWARE HARDWARE

George Eichinger *

MIT Lincoln Laboratory
Lexington, Massachusetts
george.eichinger@ll.mit.edu

Kaushik Chowdhury, Miriam Leeser

Department of Electrical and Computer Engineering
Northeastern University
Boston, Massachusetts
krc@ece.neu.edu, mel@coe.neu.edu

ABSTRACT

The FPGA is an integral component of a software defined radio (SDR) that provides the needed reconfigurability for dynamically adapting its transceiver and data processing functions. Because of the desire to process data faster and with less latency, researchers are looking at FPGA-based SDR. Our architecture, called CRUSH, is composed of a Xilinx ML605 connected to an Ettus USRP through a a custom interface board allowing flexible data transfer between them. In addition, we provide a framework that supports ease of use, independent programming on both devices, and integration with software running on the host. To demonstrate our platform we implemented spectrum sensing, a key step in determining channel availability before transmission in dynamic spectrum access networks. Spectrum sensing is implemented on CRUSH using FFTs for a 100x speedup; the complete sensing cycle is 10x faster than the same design without CRUSH. By reducing the load of transferring raw samples to the host and allowing a powerful FPGA extension for off-the-shelf devices, CRUSH enables advances in both protocol design and reconfigurable hardware targeting radio applications.

1. INTRODUCTION

There is an increasing interest in using Software Defined Radios (SDRs) for real-time processing and for more and more sophisticated algorithms. The Universal Software Radio Peripheral (USRP) [1] is a widely proliferated SDR platform used for implementing radio schemes. There is a wide variety of research using these devices with both software and hardware implementations[2][3][4]. However, real-time processing is not possible for software implementations with the USRP due to the latency of transmitting data between the host and the USRP over Ethernet. The solution is to accelerate algorithms in reconfigurable hardware that is connected directly to the USRP platform.

The USRP has an on-board FPGA, but it is small and has very little space for user functions as it already contains the logic needed to implement existing radio features. Additionally, it is written in such a way that it is difficult for a novice user to modify the code. An external FPGA board allows for more processing close to the radio front end without the constraints of using the existing FPGA. There are several different radio front ends, including the USRP, as well as different FPGA boards. New FPGA boards tend to be introduced at a faster rate than radio front ends, but both are rapidly changing over time. This project decouples high end FPGA processing from the agile radio frequency (RF) front end so that either device can be upgraded independently. CRUSH: Cognitive Radio Universal Software Hardware is a platform consisting of a USRP, Xilinx FPGA board and Custom Interface Board (CIB) as well as FPGA designs and software support to allow the easy integration and use of the platform. While CRUSH is demonstrated with an Ettus N210 and a Xilinx ML605 board, the framework is designed to easily connect to other radio front ends and FPGA cards. This means that a university could pair their USRP 2's purchased in 2008 with a brand new Xilinx KC705 FPGA development board using our Custom Interface Board (CIB) and the CRUSH concept to perform leading edge FPGA SDR research without significant cost increases.

The main contributions of this paper are:

- The CRUSH platform, which enables FPGA acceleration of cognitive radio applications using standard parts that are inexpensive, widely available, and widely used,
- A framework of hardware and software on USRP, FPGA and host so that CRUSH is easy to use, and
- The use of CRUSH to implement spectrum sensing.

Dynamic reconfiguration of the transmission and processing parameters is a defining aspect for SDRs. Our platform allows a real-time evolution of both the processing

*The Lincoln Laboratory portion of this work was sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

978-1-4673-2256-0/12/$31.00

algorithm and the software defined components, by independently altering the code for the component blocks of the system. CRUSH has the ability to integrate up to three off-the-shelf radio front ends with a single FPGA board, thereby creating a *super radio* with significantly increased bandwidth and processing capabilities. This provides an unprecedented level of flexibility at very low cost and allows new insights on the tradeoffs between operational overhead and performance gain when using multiple co-located radios.

The remainder of this paper is organized as follows: Section 2 contains background on SDR and spectrum sensing. Section 3 details the CRUSH platform in terms of its hardware, FPGA support and software. Section 4 explains spectrum sensing both in software and on CRUSH. Next in Section 5 we present the performance results of spectrum sensing on CRUSH. Finally Section 6 concludes the paper.

2. BACKGROUND

2.1. Software Defined Radio Platforms

There are several SDR platforms currently on the market for use by researchers. The most popular, and the one used in this research, is the Universal Software Radio Peripheral (USRP) by Ettus Research [1]. There are additional SDRs that have been used in research such as the Wireless Open Access Research Platform (WARP) by Rice University [5], and Networking over White Spaces (KNOWS) by Microsoft Research [6]. The SDC Testbed is a platform currently under development at Drexel University that combines WARP front ends with a Xilinx ML605 development board with the goal of creating a SDR platform [7]. Additionally, there are several commercial products that are advertised as capable of functioning within an SDR such as FPGA boards from Pentek, Lyratech, 4DSP, Acromag and Innovative Integration. These devices contain transceivers with ADCs and DACs but do not contain a software framework to implement radio functions out of the box like the USRP and WARP.

USRP. The Universal Software Radio Peripheral (USRP) is a product from Ettus Research that is used to implement various Software Defined Radio (SDR) schemes. The device has several different versions depending on the intended usage. The USRP N210 is the flagship SDR from Ettus and is used in CRUSH. It interfaces with a host computer via a gigabit Ethernet (GbE) link that is connected to a soft core on an FPGA. The device contains a Xilinx FPGA Spartan 3A-DSP3400. All of the logic and control of the system is programmed on the FPGA. The USRP contains two 14 bit 100 Mega Sample Per Second (MSPS) Analog to Digital Converters (ADC) and two 16 bit 400 MSPS Digital to Analog Converters (DAC). The device has a transmit and receive connector that is compatible with any of the USRP's daughter boards. The N210 supports up to 25 MHz of 16 bit I and Q data to the host over the GbE connection.

CRUSH decouples the FPGA and allows either device to be upgraded independently. Additionally, CRUSH allows the USRP to maintain its independent FPGA code while a user implements a hardware level SDR algorithm on the ML605.

2.2. Cognitive Radio

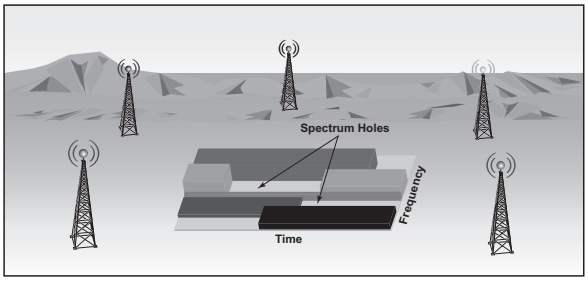

Fig. 1. Spectrum Holes Diagram

Cognitive Radio (CR) is the concept that a SDR can opportunistically utilize available spectrum when the primary user is not present. To better understand this idea we can imagine a scenario of several towers transmitting and view their spectrum representation (Fig. 1). Each tower represents one primary user (PU). A PU is someone who is licensed to use a particular band. In the center is a 3D graph with the x-axis representing time, y-axis frequency and the z-axis signal strength. Some PUs use their spectrum for all time as represented by the middle transmitter. An example of such a transmitter would be a TV station that is always broadcasting. However, if we look at the lightest station we can see that it transmits for a time, stops, then transmits again. This gap between transmissions is referred to as a spectrum hole. CRs take advantage of these holes to operate in unused portions of the RF spectrum. The detection of these holes is referred to as spectrum sensing where the radio determines which portions of spectrum are empty. This is synonymous with determining the spectrum holes from Figure 1. Spectrum sensing is crucial to the functionality of a CR because it locates possible spectrum for operation. This is what we have chosen to implement on CRUSH as the first SDR application.

2.3. Spectrum Sensing

Spectrum sensing is part of the critical path of a cognitive radio. The spectrum sensing algorithm analyzes RF data and reports what sections of spectrum are occupied and what portions are empty. There are different ways to implement spectrum sensing [2]; we implemented the algorithm using energy detection (Fig. 2). The first step receives the RF data and digitizes it via an ADC. These data are the time domain

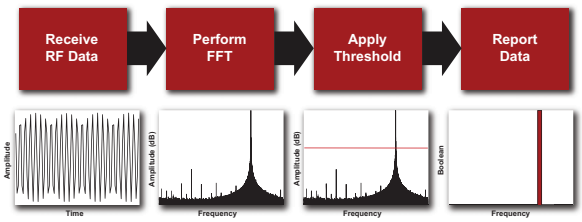

Fig. 2. Spectrum Sensing Algorithm

representation of the RF input. Next a Fast Fourier Transform (FFT) converts the data from the time domain to the frequency domain. FFTs are referred to by their point size which represents the number of output bins of the FFT. A 256 point FFT generates 256 bins, where each bin represents the energy present in a fraction of the total bandwidth. At this point the data are thresholded. If the value of the bin exceeds the threshold, the channel is occupied and assigned a value of 1. If it is below the threshold it is considered unoccupied and assigned the value 0. Once all of the FFT output values have been thresholded and assigned a value of 1 or 0, these data are aggregated and reported to the host for further processing.

3. CRUSH PLATFORM OVERVIEW

3.1. Hardware Overview

The CRUSH system (Fig. 3) is comprised of three components: (1) an Ettus Research USRP N210 software defined radio, (2) a Xilinx ML605 FPGA Development Board and (3) a custom interface board (CIB) to route signals between the USRP and the ML605. The interconnects between the boards are commercial off the shelf (COTS) cables.

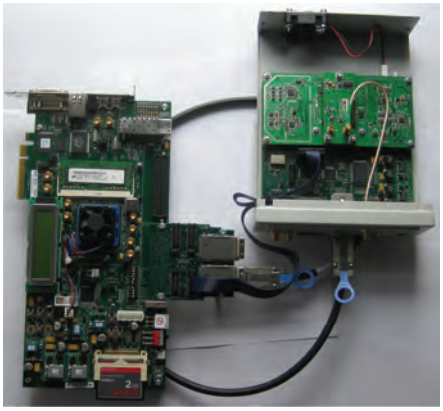

Fig. 3. CRUSH Platform

Ettus Research USRP N210. The USRP N210 (Right side of Fig. 3) is very capable but its one weakness is the FPGA. The Xilinx Spartan series is a budget FPGA that is not designed to function at high speeds and lacks advanced FPGA features compared to the Virtex series. It has less RAM and DSP blocks; features that are essential to many DSP algorithms such as FFTs and filters. Because of the complex design required to support USRP functionality, it is difficult for a user to make simple changes to this FPGA without understanding the rest of the system. Additionally, minor changes can cause the design to not meet timing closure. Modification of the existing HDL is possible but the design complexity makes this difficult.

Xilinx ML605 Development Board. To overcome the shortcomings of the USRP we added a second FPGA board. The ML605 (Left side of Fig. 3) has a Xilinx V6 LX240T FPGA which has 6.6x more Block RAM, 4.49x more LUTs and the baseline logic blocks can run ∼2.5x faster when compared to the FPGA on the USRP. The ML605 is a blank slate for the hardware developer and is not cluttered with an existing complex design like the FPGA on the USRP.

Custom Interface Board. The physical interface between the ML605 and the USRP in CRUSH is the debug port on the USRP that utilizes a MICTOR connector. To connect to this cable we created the custom interface board (CIB) (Fig. 3, center). The CIB is capable of communicating with two USRPs via a MICTOR cable and also supports the mini-iSAS USRP to USRP port labeled MIMO. The CIB is a custom PCB following the FPGA Mezzanine Card (FMC) specifications for both low pin count (LPC) and high pin count (HPC). This allows the CIB to connect two USRPs when utilized with the ML605 HPC port as well as one USRP on lower end boards with just an LPC port. The ML605 has both LPC and HPC ports and can accommodate up to three USRPs. In addition, care was taken to conform to the FMC specifications so that the CIB can function with other board and chassis manufacturers. If the ML605 is not attached to the USRP, the HDL framework has been designed so that the USRP will function as a standard radio.

3.2. HDL Overview

USRP HDL Framework. As part of the CRUSH platform, the USRP HDL must be modified to allow communication with the ML605. An overview of the USRP modules (black border), as well as the new CRUSH modules (white border) is shown in Figure 4. The first additional module required for CRUSH is the Control Bus Slave module (Fig. 4) which receives a parallel address and control data stream over the MICTOR cable from the ML605. The ML605 will send a command to select the input to the MUX. Note that data can be sent to the ML605 from two different places in the USRP processing chain as shown in Fig. 4. Once the user has selected the MUX mode, the corresponding I and Q values will be sent to the DDR Out module and then over the MICTOR

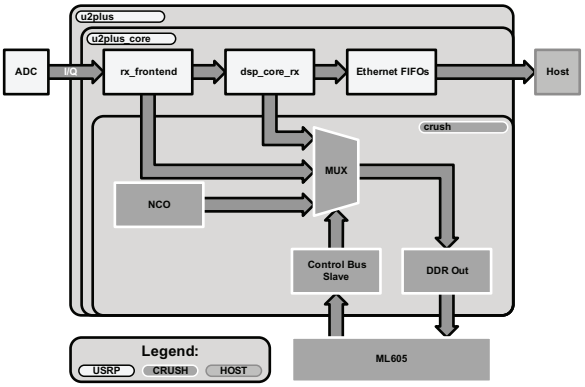

Fig. 4. USRP HDL Framework

cable.

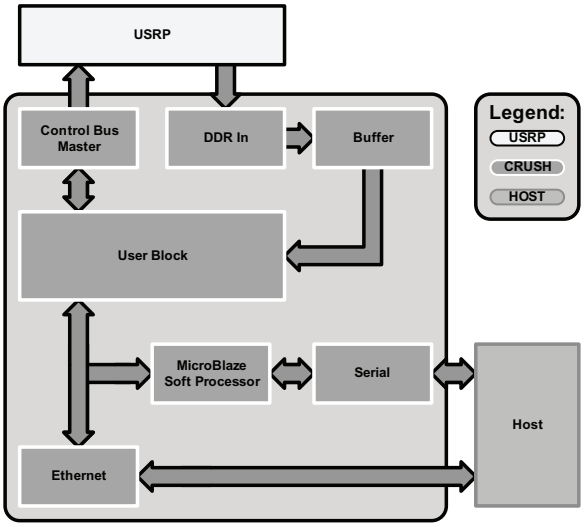

Fig. 5. ML605 HDL Framework

ML605 HDL Framework. The bulk of the CRUSH development lies in the ML605 HDL Framework (Fig. 5). The core of the ML605 HDL Framework is the User Block where the CR or SDR application resides. The glue logic like clocks, resets and wiring are in the top module along with the external interfaces. The goal of this design is to make it as easy as possible for a new user to insert their code into the project. Currently the User Block has use of 97 % of the FPGA resources after the glue logic has been implemented. A program such as Mathworks HDL coder or Xilinx AutoESL could be used to synthesize a module and insert it into the User Block. The ML605 has three different external interfaces, the MICTOR connector to the USRP, an Ethernet connection to the host and a serial connection to the host.

The MICTOR connection serves two purposes. First, it transmits data over a control bus to the USRP. Second, it receives 100 MSPS I and Q samples over a 200 MHz DDR interface. The ML605 is the Control Bus Master and provides the clock, address and control data signals. All commands that go over the control bus originate from the host via either the Ethernet or Serial port.

The Ethernet interface is based on the Xilinx Embedded Trimode Ethernet Media Access Control (TEMAC). For packets, CRUSH uses User Datagram Protocol (UDP) as the method of transport because of its low overhead and ease of use. During typical operation, the host sends the ML605 a UDP packet containing the required configuration bits and operation mode of the CRUSH system. Depending on the mode indicated by the packet, it will perform the required action and transmit the result back to the host. Ethernet packets are complex to form because they require a MAC header, an IP header and a UDP header in addition to the actual data. To quickly send packets, an embedded Microblaze processor calculates the header once and stores it to a dual ported Block RAM. Our custom Ethernet controller stores all of the possible payload lengths and corresponding checksum values so that no matter the packet length, it can load the proper checksum without recalculating.

3.3. Software Overview

CRUSH_app. The high speed code, CRUSH_app, is based on the USRP Hardware Driver (UHD). The code is written in C++ and is built on top of the UHD code from Ettus Research for communicating with the USRP over Ethernet. The code is based on the rx_samples_to_file.c example that comes with a standard distribution of the UHD software. This example was picked as the basis for our interface because it maintains all of the previous functionality of the USRP system while adding the ability to control CRUSH. UHD is based on the Boost libraries [8] and we used the UDP primitives native to Boost to implement our Ethernet interface. Using this program allows for fair speed comparison tests between CRUSH and the USRP alone since they are running identical versions of the Boost library.

Matlab Demo. On the host, we have developed a GUI demo using Mathworks GUIDE to show the functionality of the CRUSH system. The demo plots both a magnitude and log representation of the FFT values and thresholds from CRUSH. The GUI has text boxes where a user can modify the system parameters such as FFT size, center frequency, mode and threshold. The backend of the GUI contains a Matlab function that takes in the parameters from the GUI and creates a packet to interface with the ML605 HDL framework. It then opens a UDP port to CRUSH and sends the packet. When it receives the response it plots the data.

Serial Port. This connection serves as a debug port and allows a user to type commands instead of forming UDP

packets. It is useful for testing basic functionality and follows a standard command line interface. All of the functions available over the Ethernet UDP interface are available over the serial port. There are also additional debugging options that are not available elsewhere. For requests over the debug port that generate data, it will print the data values to the screen in hex. We have written a separate Matlab program that takes the values copied from the screen and generates plots. The serial port is controlled on the FPGA by the Microblaze. This yields an advantage over the Ethernet interface because the Microblaze software is coded in C and can be modified without re-synthesizing the entire design.

4. APPLICATION - SPECTRUM SENSING

The SDR application we chose to implement on the CRUSH platform is spectrum sensing (Sec. 2.3). This section describes the general algorithm and how it has been modified and implemented using the CRUSH platform.

In a traditional software defined radio without CRUSH, the ADC data are sent over Ethernet from the USRP to the host computer where they are received, an FFT is performed and a threshold applied. To implement real time algorithms, spectrum sensing requires responses in the μs range and the latency for software solutions is too slow. In the CRUSH implementation we added the ML605 so we can perform this algorithm in reconfigurable hardware. The ML605 performs a streaming FFT using the Xilinx FFT core and computes the unsigned squared magnitude of the resulting I and Q values. These values are then compared to a threshold and reported back to the user over UDP. FFT size, averaging, thresholding and scaling can be reconfigured from the host computer. Figure 6 shows a block diagram of User Block HDL that implements the above described algorithm.

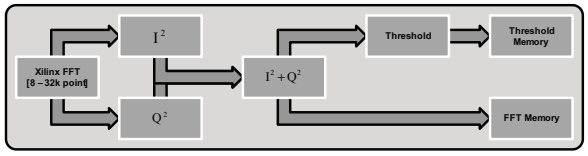

Fig. 6. User Block for Spectrum Sensing on CRUSH

When integrated into the full system, the host computer sets all of the desired parameters before starting spectrum sensing. It commands the USRP to a specific center frequency. It then tells the ML605 the threshold value for that frequency range. The ML605 immediately starts performing the spectrum sensing algorithm with the parameters given. Once done, it replies to the host with a UDP packet containing bytes where each bit represents whether a PU was found in that region. The host can then change the frequency to perform more sensing or move on to the transmit portion of the algorithm.

5. RESULTS

5.1. Functional Verification

To verify the functionality of CRUSH we tested over a range of frequencies and power levels using a signal generator connected via SMA cabling. In this paper we are reporting on one example frequency where we injected a signal into the USRP at 73 MHz with an amplitude of -16 dBm (decibels referenced to one milliwatt). We configured the device for signed 16 bit I and Q data and a total RF bandwidth of 25 MHz centered around 70 MHz. Figure 7 shows a Matlab processed 256 point FFT of the recorded data. A peak at 73 MHz is visible at 0 decibels relative to the carrier (dBc). The figure has been shifted so that the peak is displayed at value 0 dBc. By using dBc the USRP FFT can be compared to the CRUSH FFT even though they do not have the same absolute magnitude because of the effects of finite bit widths. Figure 7 is what a current software only radio would produce; we compare this to CRUSH.

We verify that our algorithm functions correctly by instructing the CRUSH platform to perform a 1024 point FFT on the incoming data and to send the raw results to the host over Ethernet. Note that the data traveling into the ML605 represents 100 MHz of data as opposed to the 25 MHz that is available to the USRP. To make the graphs more clear, Figure 8 has been cropped to just show 25 MHz. We performed a 256 point FFT on the USRP data and a 1024 point FFT on CRUSH, so that both plots show the same resolution. As with the USRP, a sharp peak is visible at 73 MHz. The peak is not exactly at 73 MHz because a 1024 point FFT will represent each bin with approximately 100 KHz of bandwidth so the boundaries do not fall on every frequency value. This is one visual example of the tests we performed to verify our system worked correctly.

Note that a peak is visible in Figures 7 and 8 at 67 MHz. This is the result of an I/Q mismatch on the analog front end of the USRP that creates a mirror image of the input around the center frequency. In this particular example the 73 MHz input signal is mirrored around 70 MHz and appears as an artifact at 67 MHz. The peak is less prominent in Figure 7 because the USRP framework decimates and filters the original data stream and only looks at 25 MHz whereas CRUSH views the stream closer to the ADC before digital filtering and sees 100 MHz. Ettus Research is aware of the issue and is working on a solution.

5.2. FFT Timing

It is difficult to measure the exact amount of time that a process takes when dealing with three systems with no time synchronization. Additionally, Ethernet packet systems are full of buffers and hard to time. We decided to make the ML605 in the CRUSH system the basis for timing analysis

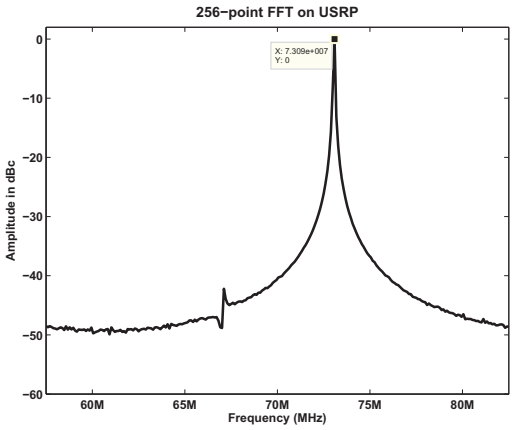

Fig. 7. USRP Test FFT Data

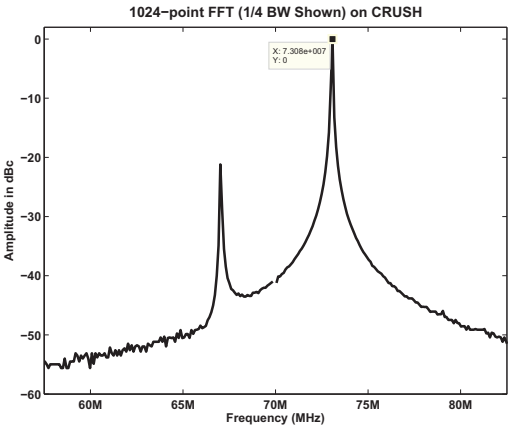

Fig. 8. CRUSH FFT Data

Table 1. Summary of FFT Timing Results

FFT Size	FPGA (μs)	Host (μs)	Speed up
8	1.17	907.72	774
16	1.91	915.89	479
32	2.38	920.07	386
64	3.56	925.47	259
128	5.47	916.54	167
256	9.56	1198.19	125
512	17.23	1003.83	58
1024	32.84	955.3	29
2048	63.55	995.26	15
4096	125.24	1071.79	8

because of the hard real-time nature of FPGAs. To record these timing measurements we inserted a timer module into the ML605 that could receive flags from various sources to record the time of events. Additionally, a module was added to the USRP to hijack the incoming data stream and insert test packets. The arrival and completion of these special packets is what yields the timing results.

To gauge the performance of the FFT completion on CRUSH, we looked at FFT sizes from 8 to 4k. For each FFT size, we ran the test 500 times and averaged the results (Table 1). We performed the analysis on both CRUSH and on the host without CRUSH but still using the USRP as the front end.

There are several interesting things to note from these results. First, the host takes almost the same amount of time regardless of FFT size. The driving factor for the completion time for the host is not the FFT, but rather the latency of transmitting the data over Ethernet. Since the spectrum sensing algorithm on CRUSH results in data compression, the results of a CRUSH FFT always fit within one UDP packet. This reduces the amount of time because the host does not need to wait for multiple UDP packets for spectrum sensing results. For the host side FFT, we are using FFTW; this is a very efficient implementation for FFTs of these sizes on a standard computer [9]. Great care was taken to optimize the host processing so that it would be a fair comparison. Even with software modifications to optimize timing, it takes approximately 6 to 7 blocks of empty data from the USRP to the host before we see the timing packet.

Table 1 shows the actual timing numbers in μs as well as the speedup between CRUSH and software. CRUSH is faster than the host because of its close proximity to the data source. Typical required FFT sizes for spectrum sensing range from 64 point to 256 point because fine resolution is not needed to determine if a PU is present [10]. For 64 and 256 point sizes we see a 259x and 125x speed up respectively. This moves the application timing from ms to μs levels; allowing CRUSH to implement near real time radio operations. In [10] the authors implement a localization algorithm based on doublethresholding and utilize a 64 point FFT taking 6 μs as compared to 3.56 μs for CRUSH.

5.3. End to End Timing

To obtain end to end timing results from a complete spectrum sensing cycle, we utilized the UHD C++ code and built a low latency Boost UDP server on top of the default software [8]. We used asynchronous send and receive threads and timed the spectrum sensing cycle using the rdtsc command. The computer was setup to minimize interference from other software such as antivirus, and the test program was run with high priority on its own processor. The main thread would start a timer and then send a packet. It would then set a mutex lock which is unlocked by the receive thread

after it has verified the packet from CRUSH.

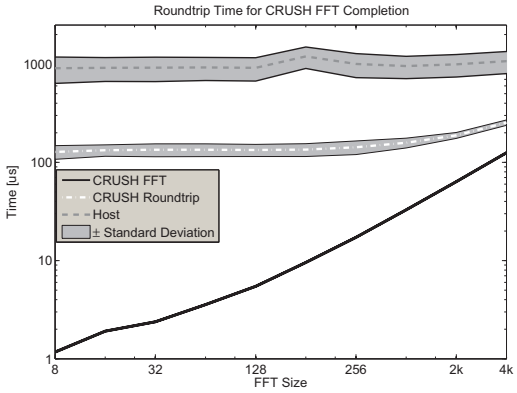

Fig. 9. Comparison of all times

For the end to end test we examined FFT sizes from 8 to 4096 and averaged over 400 packets per point size. To show these results, we combined the data from Table 1 and plotted all three measurements on the same graph (Fig. 9). The gray dashed line represents the time the host takes to complete the given FFT size. The white dash-dot line represents the end to end or roundtrip time on CRUSH and the solid black line represents the FFT completion time of CRUSH. Both the x and y axes are in log scale; the FFT size is on the x-axis and time in μs is on the y-axis. The gray filled portion represents $\pm$ one standard deviation. Since both the host and roundtrip results included UDP packet transmission times, there were large standard deviations. Analyzing this graph, we can see that CRUSH is approximately 10x faster than a host only version of the same algorithm.

After the implementation of the CRUSH framework and the spectrum sensing application that uses a simple threshold based energy detection, the ML605 still has ample space to implement additional algorithms. The CRUSH framework uses 1.2% Slice Registers, 7.8% Block RAM and 0.4% DSP48 blocks. Spectrum sensing adds 2.6% Slice registers, 26% Block RAM and 8.3% DSP48 blocks. The majority of the spectrum sensing hardware is Block RAM used for calculating the FFT. Overall with both the framework and spectrum sensing implemented, the system still has 96.1% Slice Registers, 66.3% Block RAM and 91.2% DSP48 blocks free. One candidate for using this space is to implement additional MAC functionality in reconfigurable hardware. This could lead to novel split-MAC designs, with time critical functions of the MAC performed on the FPGA, and policy decisions undertaken by the host.

6. CONCLUSIONS

We have introduced a versatile computing platform using advanced FPGA technology. CRUSH combines a powerful FPGA with the RF front end part of the SDR, transparently to the latter's operation. To enable this, we created a custom interface board (CIB) that allows high speed data transfer between two widely used COTS platforms. The signal processing is now closer to the receiver allowing for implementation of high speed, real time algorithms. This also decouples the FPGA computing resources from the SDR allowing for either to be updated independently. We implemented spectrum sensing as the first application on CRUSH. CRUSH can implement FFTs at 100x the rate of software and can perform a complete sensing cycle 10x faster than software for FFT sizes of interest to cognitive radios. We have reduced the load on the host computer by running this algorithm in hardware and we have kept the system fully configurable. The CRUSH platform supports up to three radios connected to a single ML605, which enables new avenues of research that we plan to investigate in the future.

7. REFERENCES

[1] Ettus Research. USRP - Ettus Research. [Online]. Available: http://www.ettus.com/

[2] I. F. Akyildiz, W.-Y. Lee, and K. R. Chowdhury, "CRAHNs: Cognitive Radio Ad Hoc Networks," *Ad Hoc Networks*, vol. 7, no. 5, pp. 810 – 836, 2009.

[3] T. Yucek and H. Arslan, "A Survey of Spectrum Sensing Algorithms for Cognitive Radio Applications," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 1, pp. 116 –130, 2009.

[4] MacKenzie, A.B. et al., "Cognitive Radio and Networking Research at Virginia Tech," *Proceedings of the IEEE*, vol. 97, no. 4, pp. 660 –688, april 2009.

[5] Amiri, K. et al., "WARP, a Unified Wireless Network Testbed for Education and Research," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, june 2007, pp. 53 –54.

[6] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh.

[7] Shishkin, B. et al., "SDC testbed: Software Defined Communications Testbed for Wireless Radio and Optical Networking," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on*, may 2011, pp. 300 –306.

[8] Boost. Boost. [Online]. Available: http://www.boost.org/

[9] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[10] T. Hanninen, J. Vartiainen, M. Juntti, and M. Raustia, "Implementation of Spectrum Sensing on Wireless Open-Access Research Platform," in *Applied Sciences in Biomedical and Communication Technologies (ISABEL), 2010 3rd International Symposium on*, nov. 2010, pp. 1 –5.