Linux Based Reconfigurable Platform for High Speed Ultrasonic Imaging

Spenser Gilliland and Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois 60616

Semih Aslan
Ingram School of Engineering
Texas State University
San Marcos, TX 78666

Abstract—In this paper, we present a Reconfigurable Ultrasonic System-on-Chip Hardware (RUSH) platform for real-time signal analysis and image processing. The platform is designed to directly process the full range of ultrasound from 20 KHz to 20 MHz. The project aims to make it simple to effectively develop and implement algorithms in embedded software and reconfigurable hardware. This provides the user with an opportunity to explore the full design space including software only, hardware only, and hardware/software co-design. The RUSH platform provides high speed access to a 12-bit ADC controlled by a Xilinx FPGA. Access to the ultrasound data and custom IP cores is available through a gigabit Ethernet connection managed by an embedded Linux based operating system running on a Microblaze processor instantiated in the FPGA fabric.

I. INTRODUCTION

In recent years, high speed signal analysis has been used to modernize methods of ultrasonic measurement, imaging, and testing [1][2]. It is now possible to measure object thickness when accessing only one side of a material, detect defects inside of an object, and produce images that provide a complete three-dimensional view of the internal surfaces within an object. These capabilities have given new insight into fields such as structural analysis, material science, and medical imaging. Ultrasonic technologies have continued to gain in popularity due to their high precision, lower cost and shrinking footprint.

Advances in Digital Signal Processing (DSP) are responsible for lowering the cost and shrinking the footprint of ultrasonic technologies. Instead of requiring hundreds of discrete analog components, an ultrasound measurement system can now be built using just DSP hardware, and an Analog-to-Digital Converter (ADC). The processing requirements for ultrasonic DSP algorithms have traditionally been met using Application Specific Integrated Circuits (ASIC). These devices are high performance but have substantial costs and cannot be reconfigured for different applications, thereby limiting their availability to low volume users and researchers. However, technology has progressed in the field of reconfigurable logic and many ultrasonic applications are now able to fit within the capabilities of Field Programmable Gate Arrays (FPGA). An FPGA provides all the parallelism capabilities of an ASIC while allowing the device to be reprogrammed for a given application [3]. This allows a single FPGA to be used in testing multiple algorithms that would traditionally require multiple manufacturing runs (or additional die area) to implement in an ASIC. Furthermore, FPGAs provide immediate feedback on the feasibility, performance, and validity of an implementation.

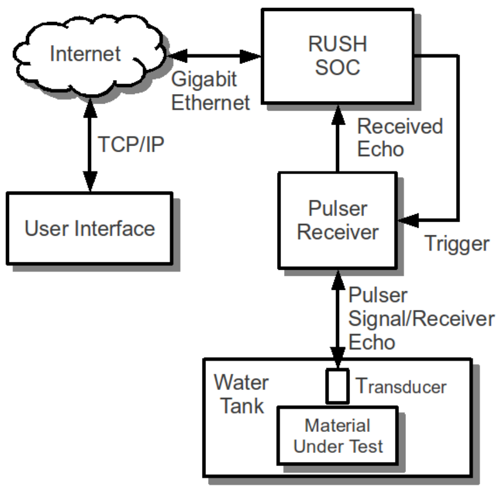

Figure 1. Ultrasound Testing Setup using a Reconfigurable Ultrasonic System-on-Chip Hardware (RUSH) Platform.

In ultrasonic non-destructive testing, properties of a material are determined by sending an ultrasonic pulse through the material and observing the reflections that occur. When the pulse reaches a discontinuity between materials, e.g. a transition from water to metal or metal to water, some of the energy from the wave is reflected back creating what is called an echo. The setup for this experiment is shown in Figure 1. In the experiment, there are exactly two dominant echoes if the material is without flaws. One occurs as the wave enters the material under test and another as the wave exits. If a flaw is present in the sample, then additional echoes will be present and various parameters of these echoes represent physical properties of the flaw.

This paper presents the design and application of a Reconfigurable Ultrasonic System-on-Chip Hardware (RUSH) platform. The RUSH platform is a collection of reconfigurable FPGA firmware and software specifically designed for processing ultrasonic signals and providing access over the public Internet.

II. RUSH OVERVIEW

The RUSH platform is designed to be an easy-to-use extensible SoC (System-on-Chip) methodology for developing low power, inexpensive, real-time processing algorithms for ultrasonic experiments. It heavily relies on existing standards to provide a common platform for both software and hardware developers.

The system takes advantage of the IP Core system in the Xilinx Embedded Development Kit (EDK). The EDK system implements a common bus based on the IBM Processor Local Bus (PLB) architecture. This allows reusable IP Cores to directly integrate with the PLB bus to provide processor, memory controller, and I/O capabilities. The major feature of the EDK tool suite is the integration of the Microblaze processor, a processor capable of running a Linux based OS. The ability to run Linux allows the reuse of a substantial portion of available software and drivers. This reduces development time and provides a common industry standard POSIX API to software developers. The ability to reuse implementations of algorithms such as TCP/IP, SSH and I2C substantially reduces development time and allows for an expanded feature set.

PLB is another common industry standard used on many IBM PowerPC based systems. It has well-understood bus timings and arbitration schemes which provide a more than capable integration point for hardware developers.

A. *RUSH Platform Architecture*

EDK provides a base system builder wizard which automatically generates a processor based system. It sets up the bus architecture and instantiates basic peripherals, e.g. DDR2 memory controller, Microblaze processor, and gigabit Ethernet controller. RUSH systems make minor adjustments to this base system to fine tune the generic SoC for ultrasound signal processing applications.

A custom designed ADC IP Core, which provides coherent averaging, integrates with the PLB bus to provide synchronous echo acquisition. When sampling is requested, the ADC core triggers the ultrasonic pulsar/receiver and starts acquiring samples after a software configurable delay. Furthermore after a software configurable pause time, the system restarts the sample acquisition process and averages the results. The number of iterations of this process is software configurable and represents the number of averages performed.

B. *Linux Implementation for the RUSH Platform*

Buildroot is used to build a Linux-based OS for a RUSH system. Buildroot provides "a set of Makefiles and patches that make it simple to generate a complete embedded Linux system." [4] Base support for the Microblaze architecture, tool chain configuration, and device tree support were added with the help of the Buildroot developers.

Because the Microblaze processor is highly configurable, certain compiler flags are required to obtain full performance from the device. We have written a program named dtopt which generates the optimum compiler flags given a device tree file. By placing these flags in the `TARGET_CFLAGS` environmental variable when building the RUSH software, the software is automatically tailored to take full advantage of the specific implementation of the Microblaze processor.

Xilinx provides an automated method for producing device tree files from an EDK project. This is accomplished through the device tree generator available from the Xilinx Git repository [5]. A device tree file is a software consumable description of the hardware devices in a SoC. This file can be either compiled into or provided to the Linux kernel at boot to ensure the kernel knows all the needed information to boot on the platform.

All hardware generics (parameters in Verilog) provided in the Microprocessor Peripheral Definition (MPD) files which are prefixed with a `C_` are available in the device tree file. This method is used to provide hardware addresses and capabilities to the drivers which implement the ADC and DAC features as well as any custom IP cores, such as SSP (please see section III). This allows the hardware and software to be more decoupled and flexible when designing. For example, the hardware engineer can increase the number of channels in the SSP algorithm without having to change any software.

The ADC and DAC cores provide access to the underlying hardware through sysfs and device file interfaces. In the default software configuration, devices can read from the ADC by reading from the `/dev/adc0` file. Parameters for the read are set by using the sysfs interface in `/sys/module/adc_plb/parameters/`. A similar scheme is used for the DAC.

Section III and section IV are application examples for the RUSH platform.

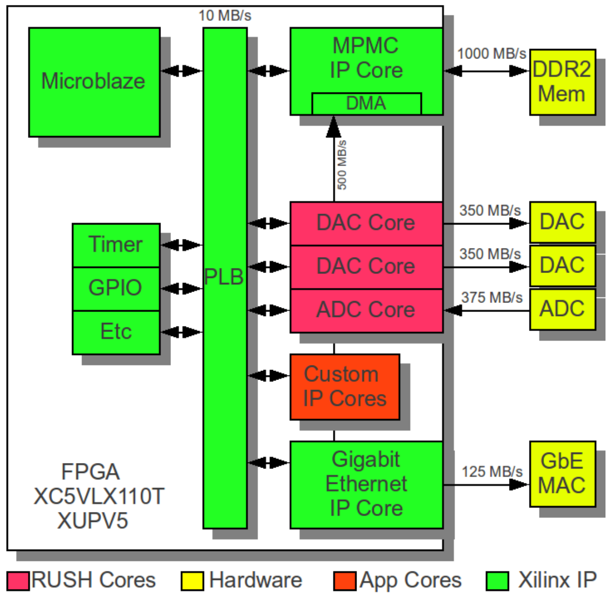

Figure 2. Firmware of the RUSH Platform

III. Split Spectrum Processing Implementation

In ultrasonic nondestructive testing, scattering echoes are generated by the microstructures of the material. These microstructures are smaller than the wavelength of the excitation wavelet, and the scattered interference patterns are highly influenced by the wavelet frequency bands. Scattering echoes can be reduced using a technique called Split Spectrum Processing (SSP) [6]. Split spectrum processing generates multiple subband signals by decomposing a broadband source (for clarification see steps 2-4 in Figure3). These subband signals are then recombined using a post processing algorithm (see Step 5 in Figure 3) which reduces the energy of the scattering echoes.

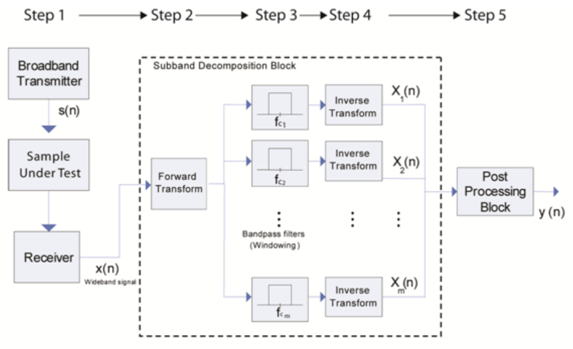

Figure 3. Split Spectrum Processing Hardware Implementation

An implementation of the SSP algorithm has been instantiated inside the FPGA fabric. The implementation is meant to be an example application for the RUSH platform. The algorithm uses basic one-zero windowing and an absolute minimum post processing block. This is a simple implementation of the SSP algorithm and methods utilizing neural networks may provide better results [1].

The hardware design of the SSP block is shown in Figure 3. The implementation makes use of the radix 2 lite FFT IP Core provided by Xilinx [7]. The radix 2 lite FFT IP core was chosen over the radix 4 or streaming FFT option due to its lower resource usage, which is critical when implementing a large number of FFTs on a single FPGA.

In Figure 4, the results from a four channel instantiation of the SSP algorithm are shown. The flaw, located at around sample #1750 is visible in all subbands where the scattering pattern is random and dependent on the frequency range of the subbands. By using a pass-through absolute minimization post processing block, the flaw maintains many of its original properties, but the scattering noise is substantially reduced.

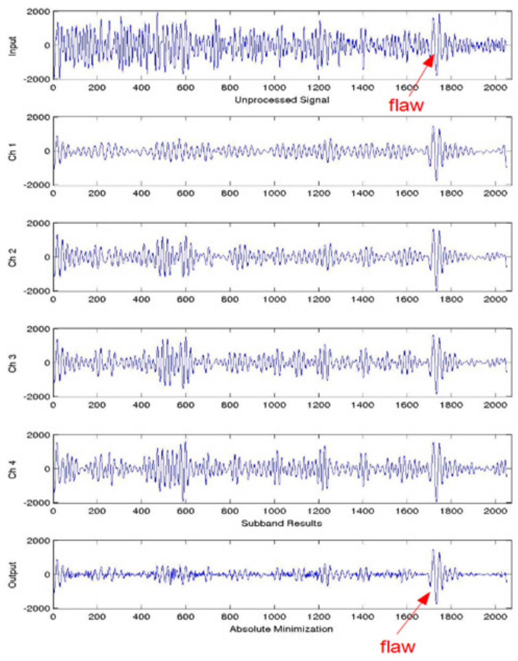

Figure 4. SSP Subbands and Results –Channel 1 covers the frequency band 0 to 4 MHz, Channel 2 covers the frequency band 0.5 to 5.9 MHz, Channel 3 covers the frequency band 1 to 5.8 MHz, and Channel 4 covers the frequency band 1.5 to 5.9 MHz.

IV. Chirplet Signal Decomposition Implementation

Chirplet signal decomposition (CSD) is a method for compressing the raw data of a reflection into a lossy format [8][9]. Because echoes are similar to chirplets and chirplets can be estimated using six parameters, the echoes can be described using six parameters. This greatly reduces the amount of data to be stored allowing for long term storage and assessment over time.

In Figure 5, the echoes from the CSD algorithm are regenerated and compared to the echoes in the original signal. Table III in the results section, shows that the SNR exceeds 10 dB when 16 estimated echoes are combined.

An implementation of the CSD algorithm was created in the C programming language. The algorithm was implemented in software due to the high complexity of the overall algorithm. The basis for the implementation is described in reference [8]; however, the algorithm has been further optimized using pre-computation and estimation methods.

The CSD algorithms main performance impediment is the correlation operation which requires regenerating a chirplet with different parameters thousands of times during each iteration of the algorithm. By pre-computing a table of these values and manipulating this table to perform the operation, the computation time was drastically reduced. Furthermore, this CSD implementation takes advantage of estimation techniques to reduce the overall computation time. This includes using lookup tables for cos, sin, and exp operations.

The Autotools build system was chosen to build the project due to its flexibility, common use among open source developers and Buildroot integration. A major benefit of using this framework was the ability to re-target the CSD application to developer machines. As the performance capabilities of the developer machines greatly exceed the performance of a

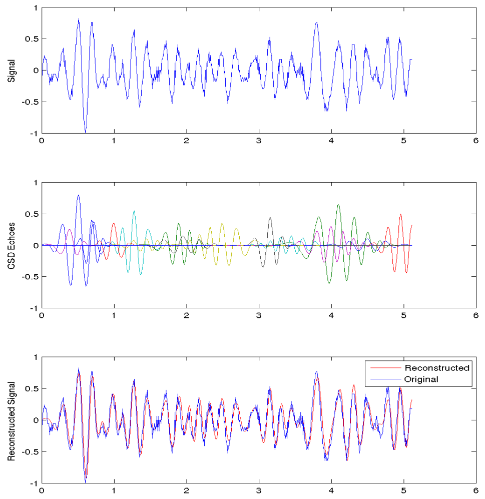

Figure 5. CSD Echo Extraction

Microblaze processor, the developers could quickly test code modifications. This was especially helpful when the code base and hardware were unoptimized and the program took over 13 hours to execute on the Microblaze processor versus about ten seconds on an Intel i7 processor.

V. Results

The total resource usage of the RUSH system is shown in Table I as well as the resource usage of the SSP algorithm with a total of 4, 8 and 12 subband channels. For each implementation, the total number of FFTs instantiated is equal to the number of channels plus one.

TABLE I. RUSH Resource Usage with SSP

Resource	*Available*	*Base*	*4 Ch.*	*8 Ch.*	*12 Ch.*
DSP48	64	6	23	41	59
SLICES	17,280	6,743	8,612	10,279	11,326
BRAMS	148	43	66	81	97

Execution times for the SSP algorithm are listed in Table II. These results show that there is no increase in execution time when additional channels are added.

The testing methodology for the SSP execution times is based on the `clock_gettime(CLOCK_MONOTONIC)` function. This function returns wall clock time in a monotonic fashion such that alterations due to network time protocol or other clock tuning would have no effect on the results.

TABLE II. SSP Execution Time for a Dataset of 2048 Samples

Channels	*Execution Time*
4	16 ms
8	16 ms
12	16 ms

Execution times for the CSD algorithm are listed in Table III. These results show a nearly linear increase in execution time with respect to number of echoes.

The testing methodology for CSD Execution times is based on the `clock()` function. This clock is only incremented during the time that the application is running. The clock was chosen because it shows the time of only the csd_read application and is not affected by other applications in the system.

TABLE III. CSD Execution Time and SNR for a Dataset of 512 Samples

Echoes	*Execution Time*	*Signal-to-Noise*
1	1050 ms	1.365 dB
2	2,050 ms	2.073 dB
4	4,040 ms	3.008 dB
8	8,060 ms	5.604 dB
15	14,880 ms	9.979 dB
16	15,940 ms	10.371 dB

VI. Conclusions

The RUSH is an adaptable system-on-chip platform for implementing various ultrasound signal analysis applications. The architecture of the RUSH system makes it simple to effectively implement algorithms using software, hardware and codesign methodologies. The base system uses approximately 39% of resources and the remaining resources can be used to implement the application specific signal processing engines.

References

[1] J. Saniie, E. Oruklu, and S. Yoon, "System-on-chip design for ultrasonic target detection using split-spectrum processing and neural networks", *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, July 2012.

[2] E. Oruklu, J. Weber, and J. Saniie, "System-on-chip subband decomposition architectures for ultrasonic detection applications", *Springer, Journal of Signal Processing,* September 2011. DOI: 10.1007/s11265-011-0623-9.

[3] J. Rodriguez-Andina, M. Moure, and M. Valdes, "Features, design tools, and application domains of FPGAs," *IEEE Transactions on Industrial Electronics*, , vol. 54, no. 4, pp. 1810 –1823, August 2007.

[4] (2012) Buildroot - usage and documentation. [Online]. Available: http://buildroot.net

[5] (2012) git.xilinx.com git. [Online]. Available: http://git.xilinx.com/

[6] J. Weber, E. Oruklu, and J. Saniie, "FPGA-based configurable frequency-diverse ultrasonic target-detection system," *IEEE Transactions on Industrial Electronics*, pp. 871–879, vol. 58, March 2011.

[7] (2012) Fast fourier transform (FFT). [Online]. Available: http://www.xilinx.com/products/intellectual-property/FFT.htm

[8] Y. Lu, R. Demirli, G. Cardoso, and J. Saniie, "A successive parameter estimation algorithm for chirplet signal decomposition," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, pp. 2121–2131, vol. 53, N ovember 2006.

[9] A. Kasaeifard, J. Saniie, and E. Oruklu, "Chirplet parameter estimator based on ellipse fitting in time-frequency distributions for ultrasonic NDE applications," *Proceedings of IEEE Ultrasonics Symposium*, pp. 2024–2027, 2010.