IET Circuits, Devices & Systems

Research Article

Architecture of the reconfigurable ultrasonic system-on-chip hardware platform

ISSN 1751-858X
Received on 27th April 2015
Revised on 25th September 2015
Accepted on 28th October 2015
doi: 10.1049/iet-cds.2015.0146
www.ietdl.org

Spenser Gilliland, Pramod Govindan, Jafar Saniie ✉

Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616, USA
✉ *E-mail: sansonic@ece.iit.edu*

Abstract: Design of ultrasonic signal processing systems requires a paradigm shift to fully utilise the benefits of recent advancements in the field of integrated circuits. It is necessary to design a standardised common platform that provides the flexibility to develop both software and hardware solutions. This enables the user to explore the full design space including software only, hardware only, and hardware/software co-design. To fulfil this purpose, the authors introduce the reconfigurable ultrasonic system-on-chip hardware (RUSH) platform. RUSH provides a common basis which significantly reduces the effort required to develop an ultrasonic signal processing system able to process the full range of ultrasound from 20 kHz to 20 MHz. Furthermore, this study aims to make the design and implementation of signal processing algorithms in embedded software and reconfigurable hardware very efficient. To demonstrate the computational efficiency and design flexibility of the RUSH platform, several important computationally intense algorithms such as split spectrum processing, chirplet signal decomposition and coherent averaging have been successfully ported to the RUSH platform, emphasising the many parts of the RUSH architecture.

1 Introduction

In recent years, ultrasonic measurement and testing applications have been using high speed signal analysis for improved diagnostics [1–5]. Ultrasonic technologies are popular due to their high precision, low cost and shrinking footprint. This allows to easily measure material properties, and to obtain high-resolution images that provide geometric characteristics of an object. Furthermore, this enables to perform a complete three-dimensional analysis of the internal structure of the object. These capabilities have given new insight into fields such as material science, structural health monitoring and medical diagnostics.

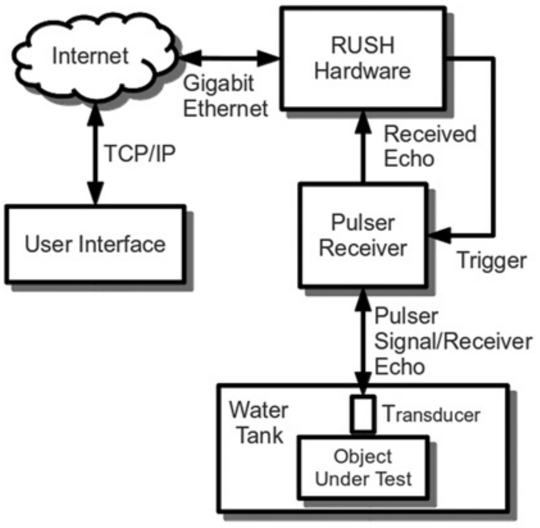
Advancements in system-on-chip (SoC) technologies have contributed to the lowering cost and shrinking footprint of ultrasonic technologies. Instead of using several discrete and analogue components, an ultrasound measurement system can be built by using digital signal processing (DSP) hardware, along with an analogue-to-digital converter (ADC). Application specific integrated circuits (ASICs) are traditionally used to meet the computational processing requirements of ultrasonic signal processing algorithms. AISCs are high performance devices, which cannot be reconfigured for various applications. Furthermore, they have considerable development and manufacturing cost. Conversely, reprogrammable logic has advanced such that most of the industrial applications are able to fit within the capabilities of field programmable gate arrays (FPGAs) [3, 6–16]. FPGA devices have been used for ultrasound [3], pulsed-wave Doppler [15], flaw detection using split-spectrum processing [6], fast chirplet-based signal processing [16], neural networks [2, 12], floating point matrix multiplication [11], industrial control systems [9, 13], micro-electromechanical system prototyping [10] and for image and video processing [14]. FPGAs allow the device to be reprogrammed for a particular application, while offering comparable parallelism capabilities of an ASIC. Therefore, a single FPGA can be used to test multiple algorithms that would otherwise require multiple manufacturing runs (or additional die area) to implement in an ASIC. Furthermore, FPGA-based implementations provide immediate feedback on the feasibility, performance and validity of an implementation, by undergoing an iterative design approach.

In practice, non-destructive evaluation (NDE) of materials requires the design of the transducer sub-system to be tightly coupled to the data collection system in order to ensure high sampling rates and real-time computation. For high-resolution analysis of materials, ultrasonic measurements involve transducers and ADCs operating in the multi-megahertz frequency range. This results in a significant computational load for embedded systems.

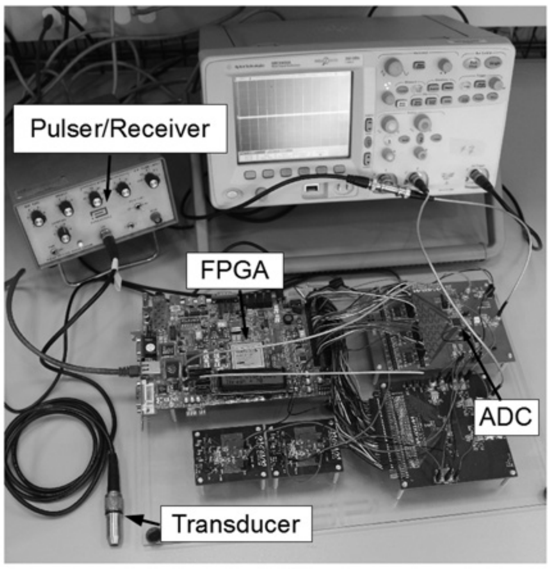
This paper presents the architecture and application of a reconfigurable ultrasonic SoC hardware (RUSH). The core contribution of RUSH is the creation of a library of FPGA firmware and Linux drivers/software to control the collection and algorithmic inspection of ultrasonic data; thus, making RUSH a suitable platform for realising ultrasound systems. RUSH integrates a powerful Microblaze embedded processor which is capable of executing platform independent *C* code including the Linux Kernel and GNU user land utilities [17]. This enables the reuse of existing software, and thus helps to reduce design and development time. In addition, RUSH supports reuse of hardware intellectual property (IP) cores, which can be easily integrated with the processor using standardised bus architectures. Furthermore, the bus architecture has been integrated into off-the-shelf operating system (OS) so that device drivers can be easily reused.

The RUSH platform offers three possible extension options for user applications. First, the user application can be easily implemented in Linux software using the processor. However, this provides low computational performance. Second, an application can be implemented using the bus architecture. This provides enhanced performance, however consumes substantial time to implement and verify. Finally, the applications requiring synchronous access to the ADC data can use the pre-processing block available within the ADC component of RUSH architecture. In addition, a combination of these three options can be used to implement a co-design methodology or use the transmission control protocol/Internet protocol (TCP/IP) stack in combination with a network connection to enable networked operations.

To evaluate the extension options of the RUSH architecture, three signal processing algorithms, each of which utilise a different design flow are implemented. Chirplet signal decomposition (CSD) algorithm is implemented as an embedded Linux application executed on Microblaze processor configured within the FPGA. A dedicated hardware solution is evaluated through the implementation of the split spectrum processing (SSP) algorithm.



a



b

Fig. 1 *RUSH platform architecture*

a Ultrasound testing setup using a RUSH platform
b RUSH system including Virtex-5 SoC FPGA, ADC, pulser/receiver and ultrasonic transducer

Finally, coherent averaging is realised by using the pre-processing block of the ADC firmware. Furthermore, the RUSH system is configured to provide high speed access to a 12-bit 200 MSPS Maxim MAX1215N ADC controlled by a Xilinx XC5VLX110T FPGA [18]. In addition, the system includes gigabit Ethernet for data acquisition.

Many systems have been developed by utilising FPGAs for ultrasound signal processing. However, most of these efforts neither support user programmability in both hardware and software, nor provides a single chip solution (ADC plus FPGA). In [2, 16, 19, 20], the authors present systems which are single purpose designs and are not created for user extensibility. In [3–7, 21, 22], the authors describe systems which are not single chip solutions. RUSH supports hardware and software programmability. Furthermore, RUSH platform can be adapted to integrate ADC and FPGA within a single SoC FPGA. Hence, RUSH uniquely offers the capabilities needed for the design of deeply embedded ultrasonic devices.

This paper is organised into four sections. Section 2 discusses the architecture of the RUSH platform and presents the detail of both the software and hardware components created. Section 3 discusses applications which have been ported to RUSH platform and their performance. Section 4 concludes the paper.

2 RUSH platform architecture

The RUSH platform [23] is designed to be a versatile system for developing high throughput real-time ultrasonic signal processing applications. As shown in Fig. 1*a*, a primary requirement is the use of an ultrasonic pulser to excite the transmitting transducer. This transducer converts the electrical signal into an acoustical radiation field. The generated ultrasonic wave both propagates and reflects echoes corresponding to the characteristic of the propagation path in the object under test. These reflected ultrasonic waves excite the receiving transducer which converts the transient vibration energy into an electrical signal. For example in NDE of materials, there may be cracks or cavities created by stress during use, impurities during manufacturing and so on. Such cracks and cavities introduce a discontinuity within materials, which can be observed using ultrasonic pulse-echo measurement techniques.

Fig. 1*b* shows the experimental platform in which RUSH system is used for NDE. This experiment is slightly different from the standard experiment due to the replacement of the oscilloscope with RUSH hardware. The RUSH hardware not only emulates the oscilloscope but also provides additional functionality and user programmability. This allows for completely automated NDE. Furthermore, a single element transducer is used in this study for prototyping purpose, however, this system can be adapted to multi-element and matrix probes as well. Using the example RUSH system utilised throughout this study, the backscattered and reflected echoes can be examined with a precision of 12 bits and a rate of up to 200 MSPS.

RUSH system consists of an FPGA and an ADC. The FPGA provides both reconfigurable logic and a microprocessor [24]. A Linux-based embedded OS built using Buildroot is installed on the processor to provide a common POSIX compliant programming environment to software designers [25]. Drivers are provided for the network controller and ADC. Hardware designers may use very high speed integrated circuit hardware description language (VHDL) or Verilog to create new hardware accelerators or implement pre-processing elements for the ADC. RUSH provides the opportunity for designers to develop applications in both reconfigurable logic and software.

There are many reasons that a designer may choose either hardware and/or software for a design. For hard real-time requirements such as those required by coherent averaging or algorithms such as SSP which include highly parallel computation, hardware provides the most efficient solution. However, many applications have highly sequential components that make a hardware implementation difficult to implement. In these cases, a co-design methodology can be adopted. In a co-design methodology, the algorithm is first described in a mostly sequential nature using a software programming language. As opportunities for hardware acceleration (i.e. high degree of parallelism) are identified, portions of the algorithm are instantiated in the FPGA firmware. As some applications do not require significant processing effort or are specifically suited to the microprocessor architecture, the application may require no hardware acceleration to meet its requirements. Under all circumstances, the resource utilisation and performance must be studied to analyse the efficiency of the implementation.

2.1 Hardware design

The example RUSH system used in this study includes a Xilinx XUPV5 FPGA development platform, a Maxim MAX1215N ADC evaluation kit, a Maxim MAX5874 DAC (digital-to-analogue converter) evaluation kit and two Maxim MAX1536 power supply evaluation kits. The interconnections of these components are shown in Fig. 2. The main goal in choosing an ADC was to design

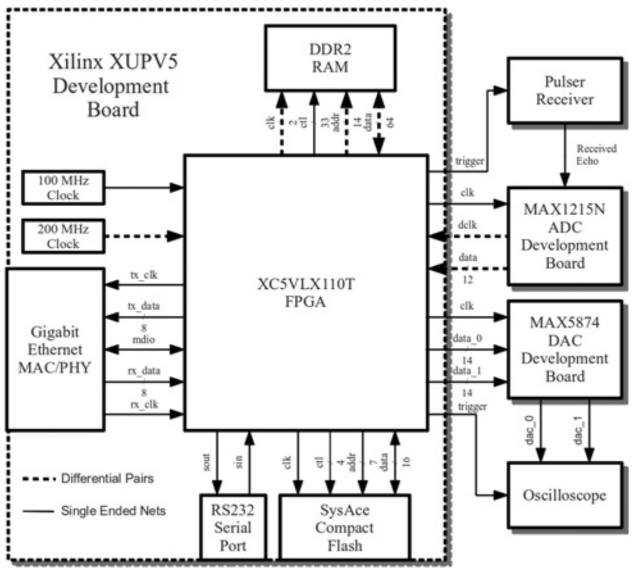


Fig. 2 *Simplified schematic of the RUSH platform*

for a broad frequency spectrum of ultrasound from 20 kHz to 20 MHz. In this study, the MAX1215N is capable of supporting up to 200 MSPS which is well above the minimum required sampling rate (i. e. Nyquist rate). This allows enough bandwidth to capture all the received ultrasonic echoes. The ADC uses low-voltage differential signalling signals which were correctly terminated inside the FPGA, ensuring proper line termination. To meet the 200 MHz clock requirement, the recovered clock feature of the ADC was implemented as shown in Fig. 3, in which, the recovered clock is aligned with the data which ensures that minimal clock skew is present (assuming equal length data/clock traces) and allows the ADC to operate at its maximum operational frequency.

By analysing the signal integrity of the inter-chip connections on this RUSH system, it is possible to ensure the correct operation of the system. This removes a critical step required in the early development of signal processing algorithms and collection systems.

2.2 FPGA firmware

The RUSH platform is carefully designed to be used as an extensible platform for the development of low power, low cost, real-time ultrasonic signal processing algorithms using SoC. By using existing industry standards, the design provides a common platform for both hardware and software researchers. Standardisation helps to re-run the previously developed experiments of our peers, and to precisely compare the performance of different algorithms.

The RUSH platform utilises the IP core system in the Xilinx embedded development kit (EDK), which allows to design an SoC around a common bus based on either the IBM processor local bus (PLB) or advanced eXtensible interface architecture. The bus timings and arbitration schemes are well-understood, which make the integration much easier for hardware designers. Reusable IP cores that are directly integrated with the bus are used to provide processor, memory controller and I/O capabilities. The main strength of the EDK tool suite is the integration of an embedded processor, which is capable of running a Linux-based OS. The ability to run Linux allows the reuse of a major portion of existing software and device drivers. This significantly reduces the system development time and provides a common industry standard POSIX API to software designers [25].

EDK provides a base system builder wizard, which automatically generates a processor-based system. It configures the bus architecture and instantiates basic peripherals such as DDR2 memory controller, Microblaze processor and gigabit Ethernet controller. The RUSH platform provides additional components to this base system in order to fine tune the generic SoC for ultrasound signal processing applications. As shown in Fig. 4*a*, the SoC FPGA integrates the following Xilinx IP cores: a multi-port memory controller (MPMC), universal asynchronous receiver transmitter, gigabit Ethernet, system advanced configuration environment compact flash controller, timer, clock generators and a Microblaze processor. These cores are interconnected via the PLB. The RUSH system communicates with the user using TCP/IP via gigabit Ethernet. To allow ultrasonic measurements, the RUSH ADC firmware is connected on the PLB bus. The design of this firmware is described in detail in the following section.

The firmware for the ADC is implemented in VHDL using the IEEE signed integer type (two's complement) and integrates with the PLB bus. A block diagram of the ADC firmware design is shown in Fig. 4*b*. The ADC firmware is highly configurable and provides the VHDL generics in Table 1 to configure the device during synthesis.

To adapt the ADC firmware to a different ADC, only the ADC front end component must be rewritten. This component transforms the ADC output into an array that includes the last C_ADC_BLOCKS samples.

In this design, the dclk net is fed into a delay element where the amount of delay for the net is decided by the place and route tool at runtime, such that it optimises the alignment between data and dclk at the first synchronous element. As it was found through experimentation that global clock routing introduces a large delay on the clock line, the relatively low fanout local clock routing is used to provide dclk routing before the regional clock buffer (BUFR) component. After the BUFR component, global clock routing is used in order to take advantage of the low skew global clock areas. This provides an aligned clock to the initial synchronous element and a good driver for the rest of the logic in the FPGA. This ensures minimal delay for the data thus making it easier to meet timing.

The samples are stored in an asynchronous dual port block RAM (BRAM). One port allows access from the adc_clk domain and another allows access from the usr_clk domain. In the adc_clk domain, the pre-processing block is able to access the current value in the BRAM and also to store new values. From the usr_clk domain, the ADC component provides a memory mapped interface to the processor. A total of ten registers are used to control the operation of the ADC. The ninth and tenth registers are used to locate the sample data BRAM interface in memory. Each sample is sign extended to a 32 bit integer because the processor is most efficient with a 4 byte alignment when operating on the integer data type. Finally, to avoid contention, a chip enable signal prevents concurrent access from both the clock domains.

2.3 Linux development

A Linux-based OS can be broken into two basic parts. The root file system is the portion which contains the initialisation scripts, configuration files and all of the common application binaries. The Linux kernel is the program initially loaded onto the hardware, which provides the ability to run the binaries located on the root file system and moderates access to hardware. The combination of a kernel and a root file system is considered a Linux OS. Buildroot is used to build a Linux-based OS for the RUSH system. Buildroot

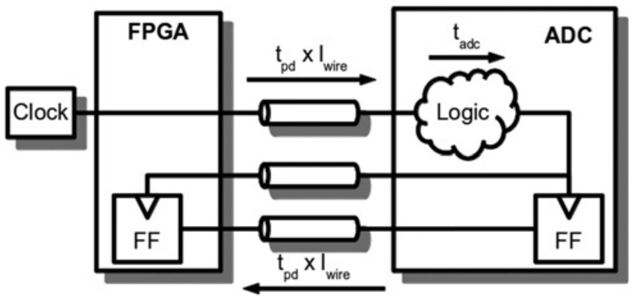


Fig. 3 *ADC to FPGA interconnection with recovered clock*

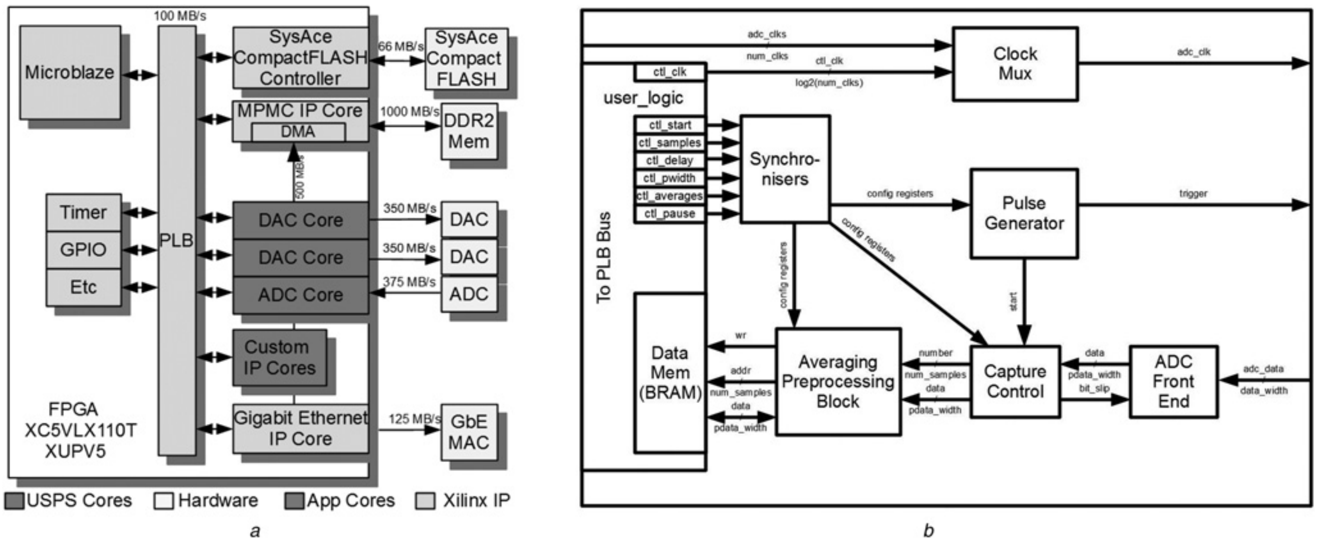


Fig. 4 *Block diagram of the ADC firmware*

a Overall firmware of the RUSH platform
b Firmware of ADC within RUSH

provides ‘a set of Makefiles and patches which helps to effortlessly generate a complete embedded Linux system’ [25]. Base support for the Microblaze architecture, toolchain configuration, and device tree support were added with the help of the Buildroot developers. Buildroot provides the software stack as shown in Fig. 5.

Linux device drivers and RUSH software libraries are added to Buildroot by adding supplementary packages for each of librush and mod_rush. Mod_rush comprises the kernel drivers for the SSP, fast Fourier transform (FFT), ADC and DAC components. Librush consists of library functions for obtaining the results from the ADC, using the SSP/FFT accelerators and some example user applications. The example user applications include adc_read, ssp_read, csd_read, and avg_read applications, which provide the ADC output in raw, SSP, CSD or averaged form, respectively. These applications read data from the device file located at /dev/adc0 directory, and interface to the character interface of the ADC driver. Furthermore, these applications use netcat or ssh to send data over the network to a receiving station. The data received can be plotted and analysed using Matlab. Matlab may be used to run comparison simulations to test hardware, aid in developing new algorithms, and to evaluate the accuracy loss of various techniques.

By using certain compiler flags, the performance of the highly configurable Microblaze processor is maximised. A device tree file

Table 1 VHDL generics for ADC firmware

Parameter	Description	Min	Default	Max
C_NUM_CLKS	Number of clocks to use with ADC	1	3	
C_ADC_BLOCKS	Amount of parallelism in the ADC frontend. NOTE: must be a power of two.	2	4	8
C_MAX_SAMPLES	Maximum number of samples to acquire.	4	4096	2^{32}
C_LOG2_DELAY	Maximum time after the pulse to wait before beginning to capture samples.	1	16	32
C_LOG2_PWIDTH	Maximum width of the pulse in samples.	1	16	32
C_LOG2_AVERAGES	Maximum number of averages in samples.	1	16	32
C_LOG2_PAUSE	Maximum delay until pulsing. Used in averaging implementation.	1	16	32
C_ADC_WIDTH	Bit width of the ADC data output	2	12	

(a software consumable description of the hardware devices in an SoC [26]) can be either compiled into or provided to the Linux kernel at boot to ensure the kernel has all the required information to boot on the platform. By using the device tree generator available from the Xilinx Git repository [27], the device trees can be produced automatically. All hardware generics (parameters in Verilog) given in the microprocessor peripheral definition files which are prefixed with a C_ are available in the device tree file. This method provides hardware addresses and capabilities to the drivers which implement the ADC and DAC features as well as any custom IP cores, such as SSP (see Section 3). This increases the design flexibility by decoupling hardware and software. For example, it is possible to increase the maximum number of samples to be obtained by the ADC without changing any software.

Drivers have been written for the various IP core devices included in the base system and are included in the Xilinx Linux kernel repository at [27]. This repository provides drivers for the gigabit Ethernet controller, MPMC, timer/counter, interrupt controller and various SoC features. In addition, the Xilinx provided toolchain was used to build the kernel and all of the binaries in the root file system.

Drivers for the ADC/DAC, SSP and FFT modules were designed and written with the help of [28, 29]. In Linux, a device can be represented by a character device, a block device or a network device. The choice to represent the ADC and SSP components as a character device was based on the belief that a character interface would be most familiar to users working on the system. A character device acts like a basic file and can be read from and written to as such. For a user, this means they can switch from a file to a real ADC with minimal changes.

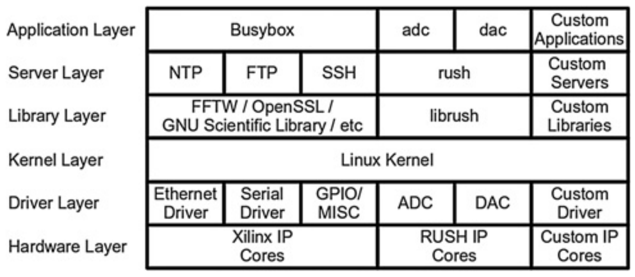


Fig. 5 *Software stack for RUSH*

3 RUSH applications

3.1 Coherent averaging

Coherent averaging is a method for improving the signal-to-noise ratio (SNR) of a pulsed signal. It is based on acquiring multiple synchronised responses and averaging them. Given an ideal signal $s(t)$ corrupted by the additive noise$v(t)$, the received signal $r(t)$ can be represented as

$$r(t) = s(t) + v(t) \tag{1}$$

The signal is assumed to be deterministic, synchronised and the same for multiple measurements. The additive noise is a random process with zero mean and noise embedded within multiple measurements which are assumed to be independent and identically distributed. Thus, the SNR of the received signal is defined as

$$\mathrm{SNR} = \frac{\langle s^2(t) \rangle}{\left[E[v(t)]^2\right]} \tag{2}$$

where $E[v(t)]^2$ indicates the expected noise power (i.e. noise variance) and $\langle s^2(t) \rangle$ represents the total signal power. Since the target is stationary and the excitation pulse is constantly repeated and synchronised, the SNR of a coherently averaged signal can be expressed by

$$\mathrm{SNR}_{\mathrm{AVG}} = \frac{\langle s^2(t) \rangle}{1/N\left[E[v]^2(t)\right]} \tag{3}$$

Inspecting $\mathrm{SNR}_{\mathrm{AVG}}$ reveals that a reduction of $1/N$ in noise power is produced by coherent averaging.

Coherent averaging heavily relies on precise synchronisation between the excitation pulse and capture logic in order to provide deterministic results for the desired signal $s(t)$. Thus, this implementation takes advantage of the pre-processing block in the ADC as shown in Fig. 6 which allows highly accurate synchronisation between the data acquisition capture clock and the excitation trigger pulse applied to the ultrasonic pulser/receiver.

Initially, a sequential averaging implementation was developed. However, it was not possible to meet timing with this original design. To explore the design space for timing closure, the original implementation was extended to support parallel operations. As shown in Fig. 6, the coherent averaging block is parallelised by a factor of four. Due to the concurrent nature of this design, the implementation is able to reliably meet the timing requirements of this system.

Since the slower distributed RAM was unable to meet the 200 MHz timing requirement, the coherent averaging block implements its memory using BRAMs. In Xilinx FPGAs, BRAMs are faster, but have a one cycle delay for access. Therefore, it takes two clock cycles for an accumulation to occur. The first cycle is a read cycle and the second is a write cycle. During the read cycle, the BRAM exports the value to a D-type flip flop (DFF). During the write cycle, the incoming data is added to the value in the DFF and stored back in the BRAM.

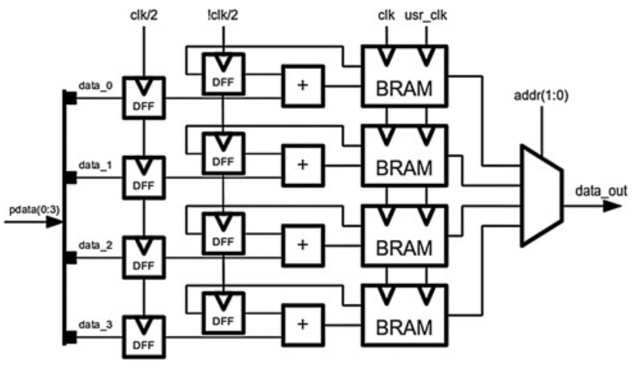


Fig. 6 *Coherent averaging implementation in real-time*

3.2 Split spectrum processing

The SSP algorithm requires significant processing resources which render it unable to complete within a reasonable deadline using the processor. Therefore, to accelerate the computation, SSP algorithm is implemented as a hardware design. The SSP block shown in Fig. 7*a* is instantiated in the FPGA fabric such that the memory element is shared between the SSP processing element and the processor local bus. The hardware implementation uses Xilinx IP cores such as radix 2 lite FFT IP core. Resource usage for radix 2 lite FFT IP core is low compared to radix 4 or streaming FFT cores. This significantly improves the resource utilisation when a large number of FFTs are implemented on a single FPGA.

The results from a four channel instantiation of the SSP algorithm are shown in Fig. 7*b*. The flaw, located close to the 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 minimisation post processing block, the flaw maintains many of its original properties, but the scattering noise is substantially reduced. Table 2 shows the total resource available within RUSH system, and the resource usage of the SSP algorithm implementations with a total of 4, 8 and 12 subband channels. For each implementation, the total number of FFTs required is equal to the number of channels plus one.

Execution times for the hardware and software implementations of the SSP algorithm are given in Table 3. These results indicate that execution time does not depend on the number of channels for hardware implementation. Moreover, a fully software implementation is at least 1000x slower, and there is a dependency between execution time and number of channels. 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.

3.3 Chirplet signal decomposition

Ultrasonic signals often consist of several interfering echoes. These echoes are similar to a chirplet, which can be represented with six parameters using CSD [16, 19]. Consequently, CSD representation significantly compresses the raw ADC data for faster data analysis and storage. Furthermore, the estimated chirplets can be used for material characterisation and system identification.

The chirplet signal can be defined as

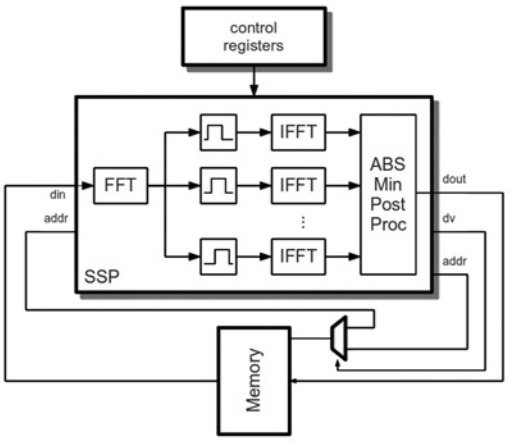
$$f_\theta(t) = \beta \exp\left[-\alpha_1(t-\tau)^2 + i2\pi f_c(t-\tau) + i\phi + i\alpha_2(t-\tau)^2\right] \tag{4}$$

Equation (4) has six parameters: $\Theta = [\tau, f_c, \beta, \alpha_2, \phi, \alpha_1]$ where τ is the time-of-arrival, f_c is the centre frequency, β is the amplitude, α_2 is the chirp rate, ϕ is the phase and α_1 is the bandwidth factor of the chirp echo. The algorithm [16] described below decomposes the chirplet signal and determines the values of the chirplet parameters to accurately model the ultrasonic echoes.

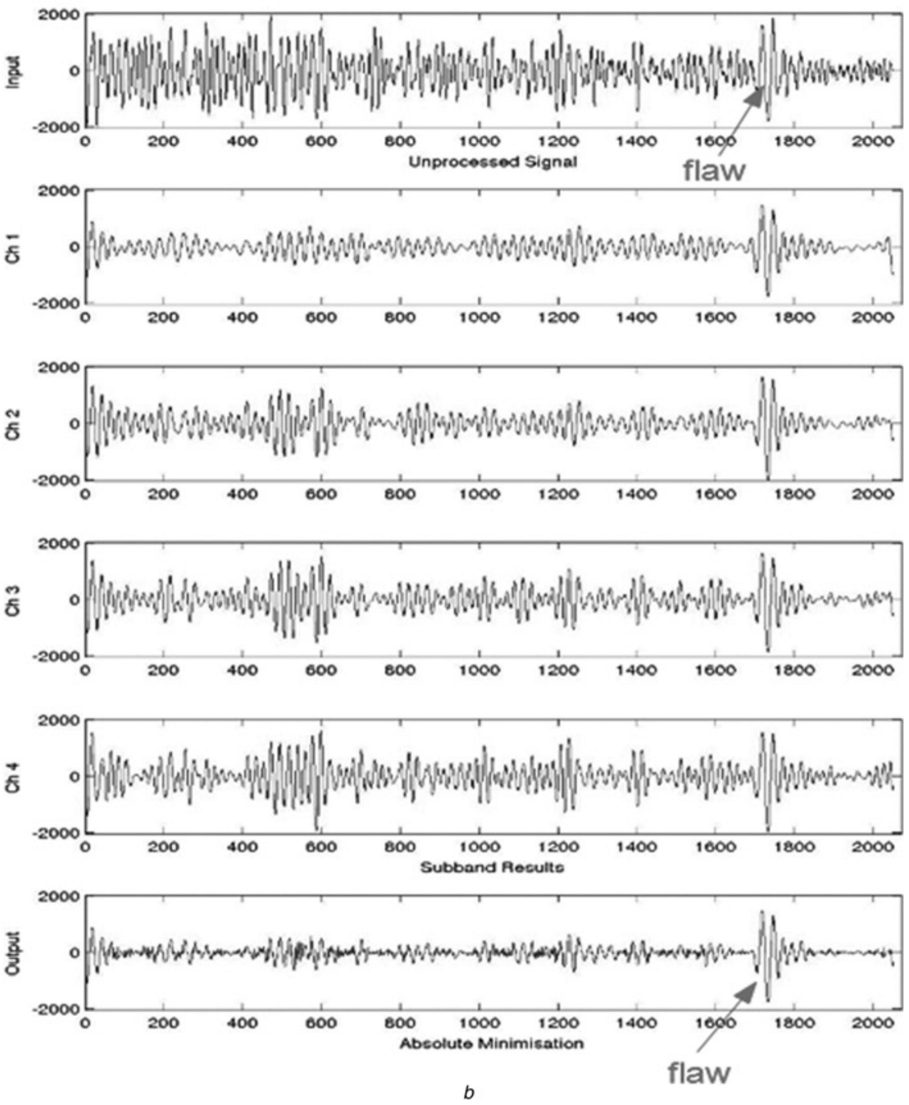
Consider a multi-component signal $s(t)$ consisting of many interfering chirplet-shape echoes

$$s(t) = \sum_i f_{\theta_i}(t) \tag{5}$$

The location of the maximum amplitude of $s(t)$ in the time domain is determined, which is considered as the initial guess of time-of-arrival. Then the centre frequency which maximises the chirplet transform is estimated, given an initial guess of time-of-arrival. Subsequently, the time-of-arrival which maximises the chirplet transform is estimated, given the estimated centre frequency from the previous step. The estimation of centre frequency is repeated until the convergence condition (if $\delta\tau < \tau_{\lim}$



a



b

Fig. 7 *SSP algorithm*

a SSP implementation

b SSP experimental results –Channel 1 covers the frequency band 0–4 MHz, Channel 2 covers the frequency band 0.5–5.9 MHz, Channel 3 covers the frequency band 1–5.8 MHz and Channel 4 covers the frequency band 1.5–5.9 MHz. (*X* axis is sample number and *Y* axis is 12-bit signed amplitude)

and $\delta f_{\text{lim}} < f_{\text{lim}}$ where τ_{lim} and f_{lim} are the predefined convergence conditions for the arrival time and centre frequency, respectively) is satisfied. Then the algorithm moves forward and successively estimates the amplitude β and the remaining parameters α_2, ϕ and α_1 using correlation for the best match between the signal and the chirplet model [see (5)]. Then the residual signal is determined by subtracting the estimated echo from the signal, and the energy of the residual signal (E_r) is calculated. The algorithm is completed when $E_r < E_{\text{min}}$, where E_{min} is the predefined convergence condition.

Table 2 RUSH resource usage with SSP

Resource	Available	Used	4 channels	8 channels	12 channels
DSP48	64	6	23	41	59
SLICES	17,280	6743	8612	10,279	11,326
BRAMS	148	43	66	81	97

Table 3 SSP execution time for a dataset of 2048 samples

Channels	Execution time	
	Hardware, ms	Software, s
4	16	22.32
8	16	39.76
12	16	56.06

The chirplet transform of the signal is represented as

$$\mathrm{CT}(\Theta) = \int_{-\infty}^{+\infty} f_\Theta(t)\Psi_\Theta^*(t)\mathrm{d}t \tag{6}$$

where $\Psi_\Theta^*(t)$ is the chirplet kernel and can be represented as

$$\Psi_\Theta^*(t) = \left(\frac{2\gamma_1}{\pi}\right)^{1/4} \exp\left(-\gamma_1(t-b)^2 - i\omega_0\left(\frac{t-b}{a}\right) - i\theta - i\gamma_2(t-b)^2\right) \tag{7}$$

CSD algorithm is implemented using C programming language. Software implementation was chosen due to the high complexity and sequential nature of the design. The above mentioned algorithm has been optimised by using pre-computation and estimation methods. Furthermore, the algorithm has been modified to optimise the parameter estimation with deterministic execution time.

The primary focus of the software implementation of the CSD algorithm is to minimise the execution time while maintaining a 10 dB SNR for echo estimation. The main time consuming operation is the correlation operations, which is associated with regeneration of chirplet with different parameters several times during each parameter estimation step. By using a pre-computed table for these correlation results, the computation time was significantly reduced. By using coarse look-up tables for sine, cosine and tangent, the execution time was considerably reduced with a slight decrease in accuracy.

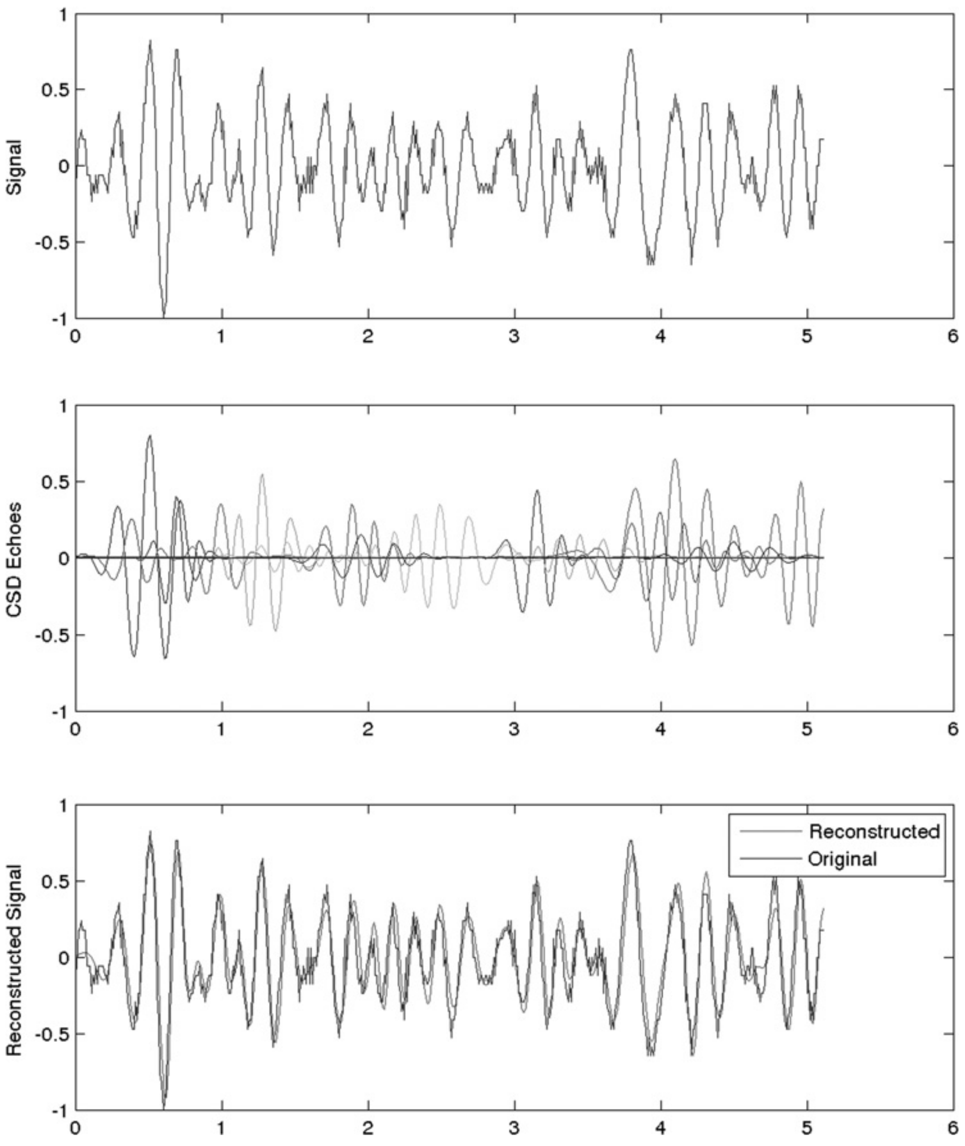


Fig. 8 *CSD for echo extraction and signal reconstruction. Top trace is the original signal, the middle trace shows the estimated echoes and the bottom trace shows reconstructed signal (grey colour) against the original signal (black colour)*

Table 4 CSD execution time and SNR for a dataset of 512 samples

Number of echoes	Execution time, ms	SNR, dB
1	1050	1.365
2	2050	2.073
4	4040	3.008
8	8060	5.604
15	14,880	9.979
16	15,940	10.371

To enforce a practical execution time, the algorithm is modified to include a finite number of iterations: only two re-estimation steps are required for time of arrival and frequency, and a total of 15 echoes are extracted. Our experimentation ensures a total SNR of 10 dB for the regenerated signal. Fig. 8 indicates that there is a high similarity between the original echoes and echoes regenerated by the CSD algorithm.

Execution times and corresponding SNR for the CSD algorithm are listed in Table 4. This shows that the execution time varies almost linearly with respect to number of echoes. Furthermore, it can be seen that SNR exceeds 10 dB when 16 estimated echoes are combined.

4 Conclusion

SoC-based platforms designed for ultrasonic signal processing applications significantly enhance the efficiency of validation and verification of any ultrasonic study. The RUSH system developed in this study is a computationally efficient and reconfigurable SoC platform for implementing various ultrasound signal processing applications. Furthermore, this study demonstrates that the flexible system architecture of RUSH allows to efficiently implement complex signal processing algorithms using hardware, software and hardware–software co-design methodologies. The integration of FPGA with a powerful embedded processor running Linux OS enables RUSH application designers to complete the design in a short span of time. Computational efficiency of RUSH system is demonstrated by implementing three practical ultrasonic signal processing algorithms: SSP, CSD and coherent averaging, at real-time rate. The resource utilisation for the base system is approximately 39% in our example RUSH system, and the remaining resources are available for implementing other application specific signal processing hardware engines.

5 Acknowledgments

The authors express their special gratitude to Xilinx Inc. for providing the necessary hardware and software tools to conduct experimentation for successfully completing this research work.

6 References

1 Saniie, J., Oruklu, E.: 'Introduction to the special issue on novel embedded systems for ultrasonic imaging and signal processing', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1329–1331
2 Saniie, J., Oruklu, E., Yoon, S.: 'System-on-chip design for ultrasonic target detection using split-spectrum processing and neural networks', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1354–1368
3 Boni, E., Bassi, L., Dallai, A., *et al.*: 'A reconfigurable and programmable FPGA-based system for nonstandard ultrasound methods', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1378–1385
4 Duck Kim, G., Yoon, C., Kye, S.-B., *et al.*: 'A single fpga-based portable ultrasound imaging system for point-of-care applications', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1386–1394
5 Qiu, W., Yu, Y., Tsang, F.K., *et al.*: 'An FPGA-based open platform for ultrasound biomicroscopy', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1432–1442
6 Weber, J., Oruklu, E., Saniie, J.: 'FPGA-based configurable frequency-diverse ultrasonic target-detection system', *IEEE Trans. Ind. Electron.*, 2011, **58**, (3), pp. 871–879
7 Oruklu, E., Saniie, J.: 'Dynamically reconfigurable architecture design for ultrasonic imaging', *IEEE Trans. Instrum. Meas.*, 2009, **58**, (8), pp. 2856–2866
8 Rodriguez-Andina, J., Moure, M., Valdes, M.: 'Features, design tools, and application domains of FPGAs', *IEEE Trans. Ind. Electron.*, 2007, **54**, (4), pp. 1810–1823
9 Sepulveda, C.A., Munoz, J.A., Espinoza, J.R., *et al.*: 'All-on-chip dq-frame based D-STATCOM control implementation in a low-cost FPGA', *IEEE Trans. Ind. Electron.*, 2013, **60**, (2), pp. 659–669
10 Chapuis, Y.A., Zhou, L., Fukuta, Y., *et al.*: 'FPGA-based decentralized control of arrayed MEMS for microrobotic application', *IEEE Trans. Ind. Electron.*, 2007, **54**, (4), pp. 1926–1936
11 Jovanovic, Z., MilutinovicV: 'FPGA accelerator for floating-point matrix multiplication', *IET Comput. Digital Tech.*, 2012, **6**, (4), pp. 249–256
12 Lin, F.-J., Hung, Y.-C., Chen, S.-Y.: 'Field-programmable gate array-based intelligent dynamic sliding-mode control using recurrent wavelet neural network for linear ultrasonic motor', *IET Control Theory Appl.*, 2010, **4**, (9), pp. 1511–1532
13 Shahbazi, M., Poure, P., Saadate, S., *et al.*: 'FPGA-based reconfigurable control for fault-tolerant back-to-back converter without redundancy', *IEEE Trans. Ind. Electron.*, 2013, **60**, (8), pp. 3360–3371
14 Desmouliers, C., Oruklu, E., Aslan, S., *et al.*: 'Image and video processing platform for field programmable gate arrays using a high-level synthesis', *IET Comput. Digital Tech.*, 2012, **6**, (6), pp. 414–425
15 Chang-Hong, H., Zhou, Q., Shung, K.K.: 'Design and implementation of high frequency ultrasound pulsed-wave Doppler using FPGA', *IEEE Trans. Ultrason. Ferroelectron. Freq. Control*, 2008, **55**, (9), pp. 2109–2111
16 Lu, Y., Oruklu, E., Saniie, J.: 'Fast chirplet transform with FPGA-based implementation', *IEEE Signal Process. Lett.*, 2008, **15**, pp. 577–580
17 'The GNU Operating System' http://www.gnu.org, accessed Jan 2013
18 'Xilinx, Virtex-5 user guide' http://www.xilinx.com/support/documentation/user_guides/ug190.pdf, accessed Jan 2010
19 Lu, Y., Demirli, R., Cardoso, G., *et al.*: 'A successive parameter estimation algorithm for chirplet signal decomposition', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2006, **53**, (11), pp. 2121–2131
20 Fritsch, C., Camacho, J., Brizuela, J.: 'A full featured ultrasound NDE system in a standard FPGA', *ECNDT Berlin Modell. Signal Process.*, 2006, pp. 1–10
21 Alqasemi, U., Li, H., Aguirre, A., *et al.*: 'FPGA-based reconfigurable processor for ultrafast interlaced ultrasound and photoacoustic imaging', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1344–1353
22 Lewandowski, M., Nowicki, A.: 'High frequency coded imaging system with RF software signal processing', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2008, **55**, (8), pp. 1878–1882
23 Govindan, P., Gilliland, S., Kasaeifard, A., *et al.*: 'Performance analysis of reconfigurable ultrasonic system-on-chip hardware (RUSH) platform', *IEEE Int. Instrum. Meas. Technol.*, 2013, pp. 1550–1553
24 Satoh, K., Tada, J., Tamura, Y., *et al.*: 'Three-dimensional ultrasound imaging using hardware acceralator based on FPGA'. in Gallegos-Funes, Francisco (Ed.) (Vision Sensors and Edge Detection, 2010), ISBN: 978-953-307-098-8, InTech)
25 'Information technology – portable operating system interface (POSIX) operating system interface (POSIX)', (2009, ISO/IEC/IEEE 9945 (First edition 2009-09-15), pp. c1–3830)
26 'Buildroot – usage and documentation', http://www.buildroot.net, accessed January 2012
27 'IEEE standard for boot (initialization configuration) firmware: 'Core requirements and practices', IEEE Std 1275-1994, pp. i–262
28 'github.com/xilinx git', http://www.github.com/Xilinx, accessed Jan 2012
29 Corbet, J.: 'Linux device drivers', (O'Reilly, Beijing Sebastopol, CA, 2005)