IET Circuits, Devices & Systems

Research Article

IET Journals
The Institution of Engineering and Technology

Hardware and software architectures for computationally efficient three-dimensional ultrasonic data compression

ISSN 1751-858X
Received on 27th February 2015
Revised on 28th May 2015
Accepted on 8th June 2015
doi: 10.1049/iet-cds.2015.0083
www.ietdl.org

Pramod Govindan, Boyang Wang, Prashaanth Ravi, Jafar Saniie ✉

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

Abstract: Ultrasonic industrial and medical imaging applications involve acquisition of large amount of volumetric data in real time. Therefore, data storage becomes critical in many current day applications which utilise ultrasound technology. Compressing the acquired data allows possessing minimal storage and also helps to rapidly transmit information to remote locations for expert analysis. The objective of this study is to design computationally efficient architectures for implementing discrete wavelet transform-based ultrasonic three-dimensional (3D) data compression algorithm on a reconfigurable ultrasonic system-on-chip (SoC) hardware platform. In this study, hardware and software architectures of the 3D ultrasonic compression algorithm are realised using Xilinx Zynq all programmable SoC. This study demonstrates that, compressing 33 MB of experimental ultrasonic 3D data into 0.42 MB (98.7% compression) requires only 84 ms for hardware architecture, and 1 min for software architecture, making both designs highly suitable for real-time ultrasonic imaging applications. Furthermore, the 3D compression is implemented by using Open Computing Language (OpenCL) targeted on Nvidia GT 750M graphical processing unit. OpenCL implementation of ultrasonic 3D compression algorithm completes the execution in <1 sec. This approach provides improved computational performance as that of hardware architecture, and comparable flexibility as that of software implementation.

1 Introduction

Ultrasound imaging techniques are widely used in both medical diagnostics and industrial non-destructive testing applications. Ultrasonic imaging has been in use since several decades as the most common medical imaging technique because it is less expensive, and non-invasive compared with other methods such as X-rays, which are harmful to the human body [1]. Technological advancements in the field of digital signal processing, semiconductor device manufacturing and embedded systems have significantly helped ultrasound researchers to design and develop portable, low-cost and computationally efficient systems, which fulfil real-time performance requirements. Portability is essential for using ultrasound systems in remote locations such as battlefields, distance places difficult to get to and construction areas. Point of care technologies and remote health monitoring have very stringent requirements on low cost and portability to provide high-quality diagnostics at real-time rate. This can be achieved only by using solid-state devices such as field-programmable gate arrays (FPGA) which are more flexible and adaptable to continually growing clinical applications [2].

Conventional ultrasound systems are not fast enough for real-time three-dimensional (3D) data acquisition and processing [3]. This obstructs the system to perform dynamic diagnostics, which has real-time nature. Recent research works demonstrate that portable and handheld imaging systems are commercially feasible, by using intelligent systems built with powerful processors [4]. Furthermore, portable and handheld 3D ultrasound system development has many challenges. One of the main challenges is the huge amount of data acquired during scanning [4]. This imposes high storage within the system, which increases the size and cost tremendously [5]. This issue can be resolved only by compressing the data without losing the required information. Another important requirement, especially for real-time applications, is the computation time for performing various data processing algorithms including compression.

The computing platforms based on reconfigurable architectures ensure a trade-off between performance and flexibility. The present embedded hardware systems, which are equipped with well-organised software applications, can provide an efficient platform to execute real-time ultrasonic imaging applications [6]. Moreover, reconfigurable hardware platforms based on FPGAs are appropriate for applications that demand low cost, low power and high flexibility [7, 8]. The embedded processing platforms comprise of system-on-chips (SoC) which integrate powerful embedded processors such as advanced RISC machines (ARM) with programmable hardware logics for high-speed signal processing. The programmable hardware logics enable implementation of computationally intensive operations, which can perform at a very high speed. The embedded processors are feature-rich to provide the overall control of the system, as well as capable of executing the signal processing algorithms for real-time applications. The latest embedded platforms are equipped with processors in a multicore configuration. Parallel processing capabilities of multicore processor systems allow achieving higher computational speed without increasing the cost and power consumption [9]. Microprocessors such as ARM and NIOS II allow acceleration of time critical signal processing algorithms by using special custom instructions [10–12]. In this study, ultrasonic 3D data compression algorithm using discrete wavelet transform (DWT) is implemented on a Xilinx Zynq-7020 all programmable SoC platform, which comprises of ARM processor-based processing system and Xilinx programmable logic. DWT has proven to be very successful in the implementation of various signal and image processing algorithms such as image coding using compute unified device architecture (CUDA) [13], image/video compression and biomedical imaging using VLSI techniques [14, 15]. This paper utilises the high compaction properties of DWT for the compression of ultrasonic radio frequency (RF) signals, maintaining very high data compression rate with the objective to extract key features of the signal such as intensity and arrival time with a high level of accuracy [16].

The Zynq processing system contains dual-core ARM Cortex-A9 processor, cache memories, on-chip memory, external memory

interfaces, direct memory access (DMA) controller and input–output peripherals [17]. Zynq platform enables system-level differentiation, integration and flexibility through hardware, software and input/output (I/O) programmability. The ARM processor communicates with the programmable logic on the Zynq SoC through several advanced extensible interface (AXI) buses [18, 19]. Furthermore, it has been proven that ARM processor on Zynq platform outperforms other processors such as Microblaze, LEON and PowerPC for image processing applications [19]. Processor-centric approach of the Zynq platform allows the ARM processor to manage overall control of the embedded system. This provides flexibility to the user in optimising the system to meet various application requirements.

To further enhance the flexibility along with high performance, 3D compression algorithm is implemented using Open Computing Language (OpenCL) on a graphical processing unit (GPU). Real-time applications, which require complex time-consuming computational operations, can be efficiently executed by using parallel processing capabilities of GPU [20]. GPUs are used in many low-cost devices which are based on multicore systems with a complex memory hierarchy. These platforms are used as parallel coprocessors, which can execute a large number of threads in parallel. Due to the increased computing capabilities of GPUs, they have been used as accelerators in applications such as scientific simulations, computer vision, bioinformatics, cryptography and finance [21]. There are two methods to program the GPU: CUDA and OpenCL. In this study, we are using OpenCL, because CUDA is limited to Nvidia GPUs, whereas OpenCL offers a generic programming model that supports the execution on a wide variety of heterogeneous platforms [22]. OpenCL allows designers to generate simple C programs that can be interfaced to GPU to enable high level of parallel processing for improved computational performance [23]. An OpenCL application consists of two distinct parts: the host program and a collection of several kernels. The host program runs on the host such as a computer central processing unit (CPU), and kernels execute the pre-defined computations on the OpenCL compute devices. Kernels are typically simple functions that transform input memory objects into output memory objects. A kernel program executes several code lines over multiple parallel threads, by using the concept of single instruction multiple threads. Thus, a single instruction is executed from multiple threads with different input data [24]. The smallest computing element within a kernel is defined as work item. Multiple work items are grouped into local work groups. The overall work space is put into a multi-dimensional index space according to the algorithm that the user defines. The work items that are divided into the same work group will be executed in parallel [23].

Section 2 describes ultrasonic 3D data acquisition and 3D compression algorithm using DWT. Section 3 explains two types of design architectures of ultrasonic 3D data compression algorithm implemented on Xilinx Zynq all programmable SoC platform: (i) hardware implementation using programmable logic and (ii) software implementation using ARM processor. Section 4 discusses the implementation of 3D compression algorithm using OpenCL and the performance analysis of GPU implementation compared with the CPU implementation. Other existing DWT implementations are focused on image/video compression where DWT is applied on frames, whereas our study offers a unique approach to design and implement ultrasonic RF signal compression by applying DWT on the reflected ultrasonic echo signal. Since the application requirements and the characteristics of input data are different in both cases, a workable comparison between the results of existing DWT implementation with results of our study is not feasible, and thus not discussed in this paper. Section 5 concludes the paper.

2 Ultrasonic 3D data compression

Processing of large amount of volumetric data is a necessity in modern medical diagnostic and industrial applications. Therefore, data acquisition and subsequent compression becomes a major responsibility of ultrasonic development systems. Fig. 1*a* shows the block diagram of ultrasonic scanning and testing setup. In this study, a 5 MHz, 0.375 inch diameter ultrasonic broadband transducer (A3062) along with a pulser/receiver model 5052 PR is used to acquire 3D block of data from a steel block specimen with microstructural defects. This experimental data consisting of a volumetric image of 128×128×2048 samples (33 MB when each sample is represented using 1 byte) is generated using a 2 × 2 inch surface of the steel block specimen. Two stepper motors are used for scanning. The measurement points by observing at the top surface of the steel block are shown in Fig. 1*b*, where the scanning in the spatial directions (y and z) is also indicated. The acquired volumetric data consists of interfering echoes [25] representing microstructural scattering of materials. Each echo signal called as amplitude scan (A-scan) is a measurement point which has 2048 samples in x-direction as shown in Fig. 2*a*. As can be seen from Fig. 1*b*, 128 A-scans were taken per line in y-direction. A-scans were acquired on 128 such lines in z-direction. Between each A-scan in a line, seven steps were advanced by using the stepper motor. Each step is equivalent to 1/400 inch. Therefore, seven steps (i.e. the distance between two consecutive A-scans) are equivalent to 0.44 mm. Measurement points are very close to each other (0.44 mm apart) to ensure no information is missed out within the specimen under test. At the same time, the nearby measurement points will have a lot of similarity. This similarity can be utilised for compressing the data. In this study, the original 3D block of data as shown in Fig. 3*a* is compressed by the ultrasonic development platform (see Fig. 1*a*) using successive 1D compressions on x, y and z directions [16]. The compressed data is shown in Fig. 3*b*. Consequently, the compressed data can be rapidly transferred to remote locations via internet.

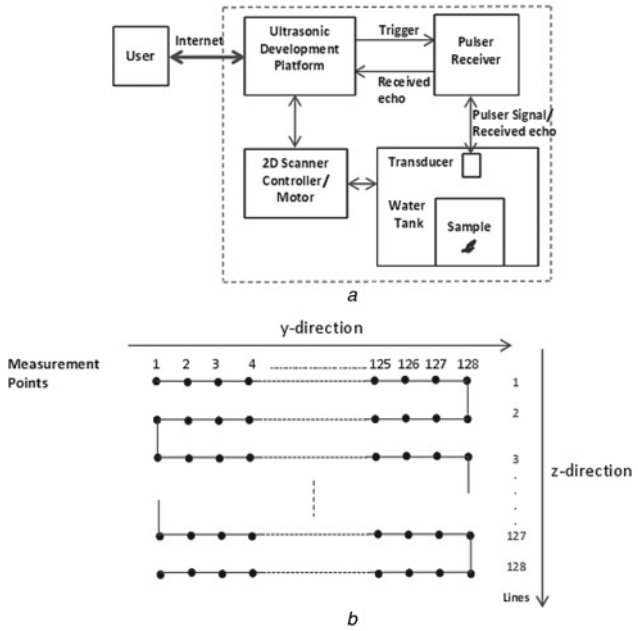

Fig. 1 *Block diagram and scanning pattern*

a Block diagram of ultrasound scanning and testing setup
b Scanning pattern of the transducer

In this study, the ultrasonic 3D data compression is performed by utilising the high energy compaction property of DWT. Multistage sub-band decomposition scheme employed within DWT [26] is carefully structured so that the high energy sub-bands, which carry most of the signal information, are isolated to obtain maximum signal compression with high-quality signal reconstruction. DWT is found to be efficient for ultrasonic time–frequency analysis and flaw detection, due to the flexibility in choosing the best wavelet kernel according to its performance for a particular application [27]. Consequently, choosing a wavelet kernel which is highly

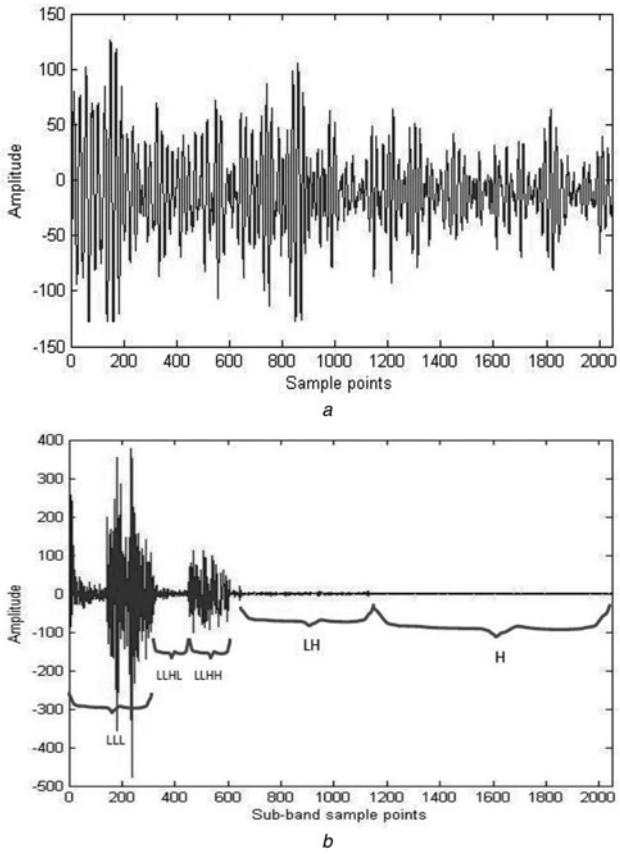

Fig. 2 *Ultrasonic A-scan and four-level DWT coefficient plot*

a Ultrasonic A-scan with 2048 samples with the sampling rate of 100 MHz using 5 MHz ultrasonic transducer
b Four-level DWT coefficient plot within sub-bands of the A-scan

suitable for decomposing ultrasonic signal ensures maximum compression. On the basis of our tests with different wavelet kernels applied on ultrasonic experimental data, daubechies-10 (Db10) wavelet kernel is chosen to decompose and maximally compress the A-scans, without losing relevant signal information [16].

A four-level wavelet packet decomposition [28, 29] structure as shown in Fig. 4 is designed to isolate the high energy frequency sub-bands in order to achieve maximum compression in x-direction. Since the compression in y and z directions will not heavily impact on the overall 3D compression ratio, a simple Haar wavelet is used for decomposition in y and z directions to maintain high computational performance. Our experimentations demonstrate that, after the decomposition in x-direction, the sub-bands H, LH and LLHL with very low energy (see Fig. 2*b*) can be eliminated [30]. The remaining sub-bands (LLL with 256 samples and LLHH with 128 samples) contain only 20% of the total signal samples. This provides 80% compression in x-direction. In y and z directions, 96 low energy sub-bands out of 128 total sub-bands are eliminated [16]. Thus, a supplementary 75% compression is accomplished in y-direction and a further 75% compression in z-direction. Therefore, the overall 3D compression becomes 98.7%. This indicates that only 1.3% of the total 3D volumetric data needs to be retained and transferred to remote locations, from which the whole 3D block of data can be reconstructed back with a very high degree of similarity.

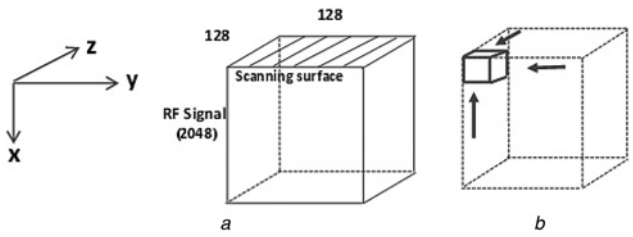

Fig. 3 *Original and compressed 3D data*

a Original 3D ultrasonic data block
b Three-dimensional compressed data

3 Three-dimensional compression on reconfigurable hardware platform

The ultrasonic 3D compression algorithm using DWT is implemented on a Xilinx Zynq-7020 all programmable SoC FPGA, which embeds a dual ARM processor unit [31]. The ARM processor unit contains two 666 MHz ARM Cortex A9 processor cores with NEON™ coprocessors that are connected in multi-processor (MP) configuration. The NEON coprocessor media and signal processing architecture consists of single instruction multiple data instructions that target signal processing applications such as audio, video and 3D graphics.

In the Zynq SoC, the programmable logic is tightly integrated with the ARM processor unit via high-speed AXI interface [31]. Moreover, the DMA capability facilitates faster data transfers between the processor and double data rate (DDR) memory. This configuration highly benefits in improving the computational performance of 3D ultrasonic data compression implemented on the Zynq platform.

Two implementation methods of the 3D compression algorithm are examined for this study: (i) hardware design using programmable logic and (ii) software design using ARM processor. DWT primarily involves filtering operations, which include several multiplications. For both methods, filtering is performed by convolving the signal samples with the Db10 wavelet coefficients for the x-direction, and Haar wavelet coefficients for y and z directions.

3.1 Hardware implementation

In the hardware design, lowpass and highpass filters in each stage of decomposition are processed in parallel to improve the overall computational load. Since we eliminate the coefficients H, LH and LLHL, we require only three lowpass filters and two highpass filters organised as highlighted in Fig. 4. In this study, polyphase filters [32] are used as an alternative to conventional filters for improving the computational efficiency and hardware resource utilisation. As shown in Fig. 5*a*, the polyphase architecture operates concurrently on the odd samples $[X(2k+1)]$ and the even samples $[X(2k)]$ of input signal. The odd and even filters are processed separately by two sub-filters, which require only half the coefficients compared with conventional filter. Therefore, the filtering operation requires only half the computational time compared with conventional filters.

Since the filtered output needs to be sub-sampled, we can eliminate either the even or odd sample output. In this study, we have eliminated the odd samples $[Y(2k+1)]$. As shown in Fig. 5*a*, the H_{even} filter applied to the odd inputs and the H_{odd} filter applied to the even inputs (shown in dashed lines) are eliminated. Therefore, the filtering operation requires only half the number of resources in comparison to conventional filters.

During signal reconstruction, the compressed data is interpolated by inserting zero at alternate sample points, before filtering operation is performed. Thus, the odd samples $[Y(2k+1)]$ will always be zero, as shown in Fig. 5*b*. Therefore, the lower section (one set of odd and even filters shown in dashed line) of the polyphase architecture is not required, which further reduces the hardware resource utilisation.

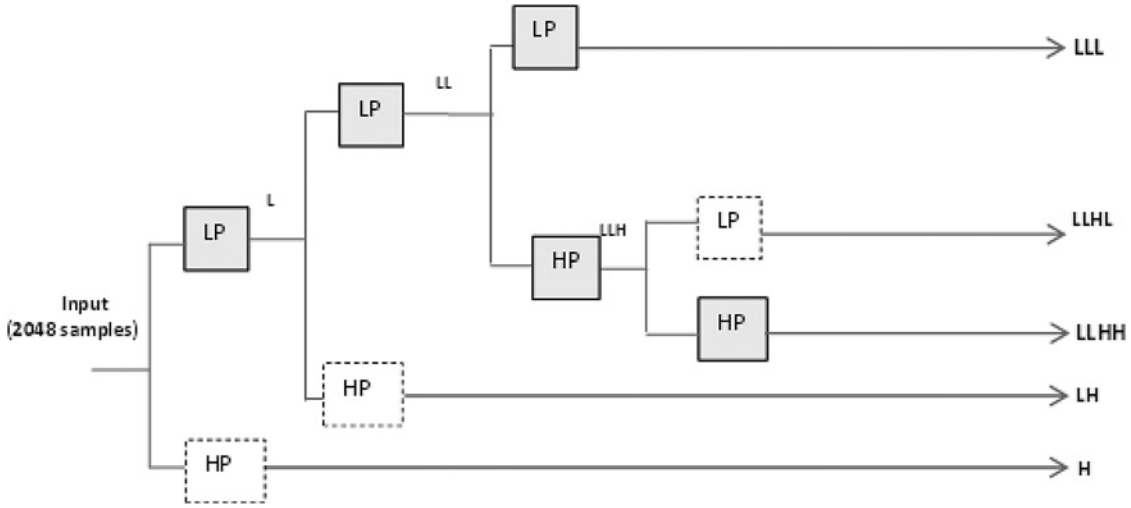

Fig. 4 *Wavelet packet decomposition in x-direction (since only LLL and LLHH are retained, the filters in dashed line are not required in the implementation)*

3.2 Hardware design optimisation and performance evaluation

The main operation in DWT is filtering. Instead of using conventional filters, data broadcast filter structures [33] are used in this study for improved computational performance. As shown in Fig. 6*a*, conventional Db10 filter has a long critical path (one multiplier and 20 adders), whereas for a data broadcast Db10 filter, the critical path has only one multiplier and one adder as shown in Fig. 6*b*. It is to be noticed that in broadcast filter, the coefficient order has been reversed.

The multiplication operations in the filters are implemented by using SHIFT and ADD/SUB operations. The operands are represented as binary numbers. To further reduce the execution time and the area of the multipliers, canonical signed digit (CSD) code is used to represent the filter coefficients [34]. CSD code uses 0, 1 and −1 to represent the binary code. It means that we can have the minimum SHIFT and ADD/SUB operations if the filter coefficients are represented in CSD form. For example, the coefficient 0.5272 is represented as 0.10000110111110 in binary, and 0.1000100**1**000**1**01in CSD. The bold '**1**'s represent minus one. Therefore, only five SHIFT and four ADD/SUB operations are needed to implement the multiplication, instead of eight SHIFT and seven ADD/SUB operations in case of binary representation. Table 1 shows a comparison between binary and CSD representations of the Db10 lowpass filter coefficients used for decomposition.

From Table 1, it can be understood that by using CSD representation, the number of SHIFT and ADD/SUB operations for conventional filters are reduced from 92 to 63, which indicates 32% reduction in number of arithmetic operations. It can also be observed that, for broadcast filter implementation, the critical path is reduced from 11 SHIFT and 10 ADD/SUB operations (coefficient −0.2498) to 5 SHIFT and 4 ADD/SUB operations (coefficients −0.1959, 0.6885, 0.5272, 0.1882) when using CSD representation, which indicates 65% improvement in computational load.

A four-level decomposition is considered for the evaluation of computational load. Furthermore, the lowpass and highpass filters are processed in parallel. Since the polyphase Db10 filters have only ten delay elements, the overall latency of the four-stage decomposition structure τ_1 becomes: $\tau_1 = (10 + 10 + 10 + 10) = 40$ cycles. Thus, the computation time to decompose one A-scan consisting of 2048 samples becomes $2048 + 40 = 2088$ cycles. Assuming pipelined data broadcast filter structures, the overall compression in *x*-direction (decomposition of $128 \times 128 = 16{,}384$ A-scans) consumes $16{,}384 \times 2048 + 40 = 33{,}554{,}472$ cycles. By performing the decomposition of two A-scans in parallel, the overall computational time in *x*-direction becomes 33,554,472/2 = 16,777,236 cycles. Similarly, by performing the decomposition of four A-scans in parallel, the overall computational time in *x*-direction becomes 16,777,236 /2 = 8,388,618 cycles.

The compression in *x*-direction eliminates 80% of the samples, which makes the 3D block size to $410 \times 128 \times 128$. This will be reduced to $410 \times 32 \times 128$ after the compression in *y*-direction. This is further reduced to $410 \times 32 \times 32$ after compression in *z*-direction. The spatial (*y* and *z* directions) compressions use a two-stage decomposition structure [16]. Furthermore, this uses

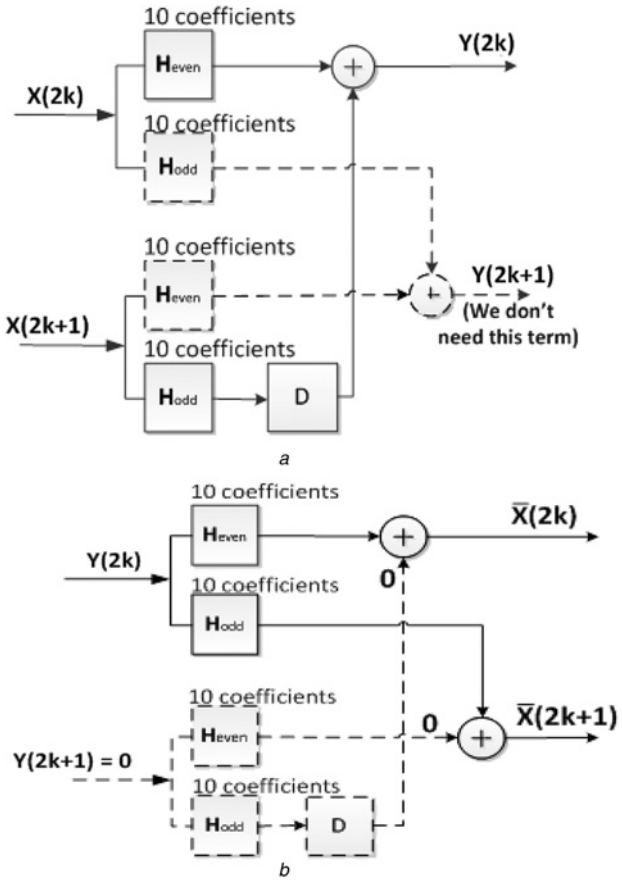

Fig. 5 *Polyphase filter [32] used for*

a Decomposition
b Reconstruction
Filters shown in dashed line are not required in the implementation

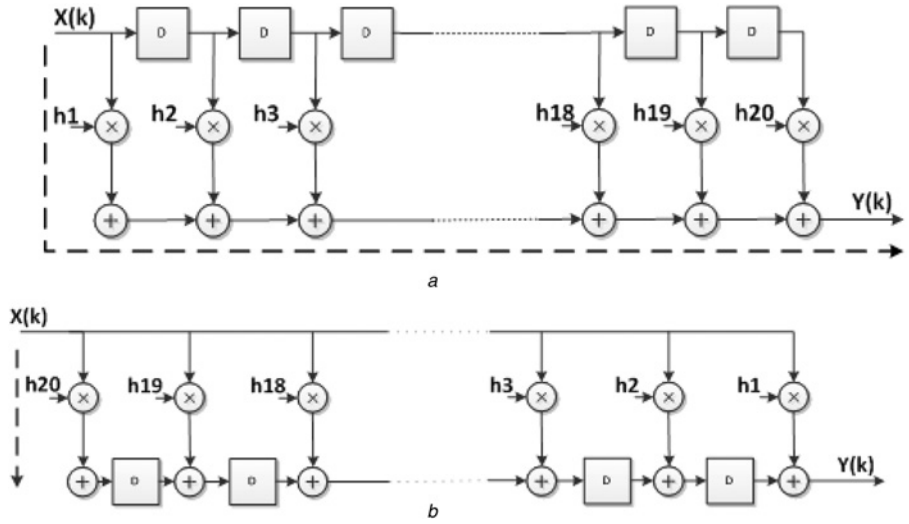

Fig. 6 *Db10 filter*

a Conventional structure
b Data broadcast structure [33]
Dashed lines indicate critical path

Table 1 Number of arithmetic operations required when Db10 lowpass filter coefficients are represented in binary and CSD

	Coefficients									
	−0.0000	0.0001	−0.0001	−0.0007	0.0020	0.0014	−0.0107	0.0036	0.0332	−0.0295
binary	0	1	1	3	1	3	**6**	**5**	2	**5**
CSD	0	1	1	3	1	3	**4**	**3**	2	**3**
	Coefficients									
	−0.0714	0.0931	0.1274	−0.1959	−0.2498	0.2812	0.6885	0.5272	0.1882	0.0267
binary	4	**7**	4	5	**11**	**9**	**7**	**8**	5	**5**
CSD	4	**4**	4	5	**3**	**3**	**5**	**5**	5	**4**

Table 2 Computational time for 1D compression in each direction

Compression	computational time (number of cycles)
X (no parallel)	33,554,472
X (two A-scans in parallel)	16,777,236
X (four A-scans in parallel)	8,388,618
Y	6,717,442
Z	1,679,360

Haar wavelet, which has only two coefficients. Thus, the latency for this decomposition scheme becomes 2. Therefore, the overall computation time in *y*-direction becomes $(410 \times 128 \times 128) + 2 = 6{,}717{,}442$ cycles. Similarly, the overall computation time in *z*-direction becomes $(410 \times 32 \times 128) + 2 = 1{,}679{,}360$ cycles.

Table 2 summarises the computational time (in number of cycles) required for three axial compressions and overall 3D compression.

Table 3 shows the computational time for 3D compression for three different approaches based on the number of A-scans processed in parallel. Analysis of the hardware synthesis timing report generated from our implementation indicates that the four-stage decomposition structure can execute at a maximum frequency of 215 MHz. Thus, we choose 200 MHz as a reasonable approximation for the calculation of computational time. Table 3 indicates that by processing four A-scans in parallel, the overall computational time has been reduced by a factor greater than 2.

Table 3 Overall computational time for 3D compression

Number of A-scans compressed in parallel	Computational time for overall 3D compression	
	Number of cycles	Milliseconds (assuming 200 MHz clock)
1	41,951,274	210
2	25,174,038	126
4	16,785,420	84

The resource utilisation for the hardware implementation of compression and reconstruction algorithm for the Zynq programmable logic is shown in Table 4.

Table 4 shows that it is quite reasonable to develop the four-stage decomposition structure with four A-scans processed in parallel, since it consumes only 56% of the look-up tables (LUT) available in the Zynq programmable logic. This confirms that the 3D compression can be completed in 84 ms, as indicated in Table 3.

3.3 Software implementation

Software implementation utilises the increasing computational power of the advanced embedded processors which are integrated within FPGA fabric [35]. Furthermore, software design allows the developers to use existing optimisation techniques to improve the system performance.

The software implementation of the 3D compression algorithm employs MP capability of ARM processor unit within Zynq to improve the computational performance. Furthermore, the parallel processing capabilities of OpenMP [36] is utilised to improve the performance of the compression algorithm execution in software. OpenMP uses specific compiler directives and flags to produce an output which is compatible for multicore processing. Moreover,

IET Circuits Devices Syst., 2016, Vol. 10, Iss. 1, pp. 54–61

Table 4 Resource utilisation of compression algorithm implemented on Zynq programmable logic

Resource	Available	All A-scans processed sequentially		Two A-scans processed in parallel		Four A-scans processed in parallel	
		Utilised	%Utilisation	Utilised	%Utilisation	Utilised	%Utilisation
flip-flop	106,400	3365	3	6730	6	13,460	12
LUT	53,200	7477	14	15,432	29	30,864	56
BUFG	32	6	19	11	34	12	38

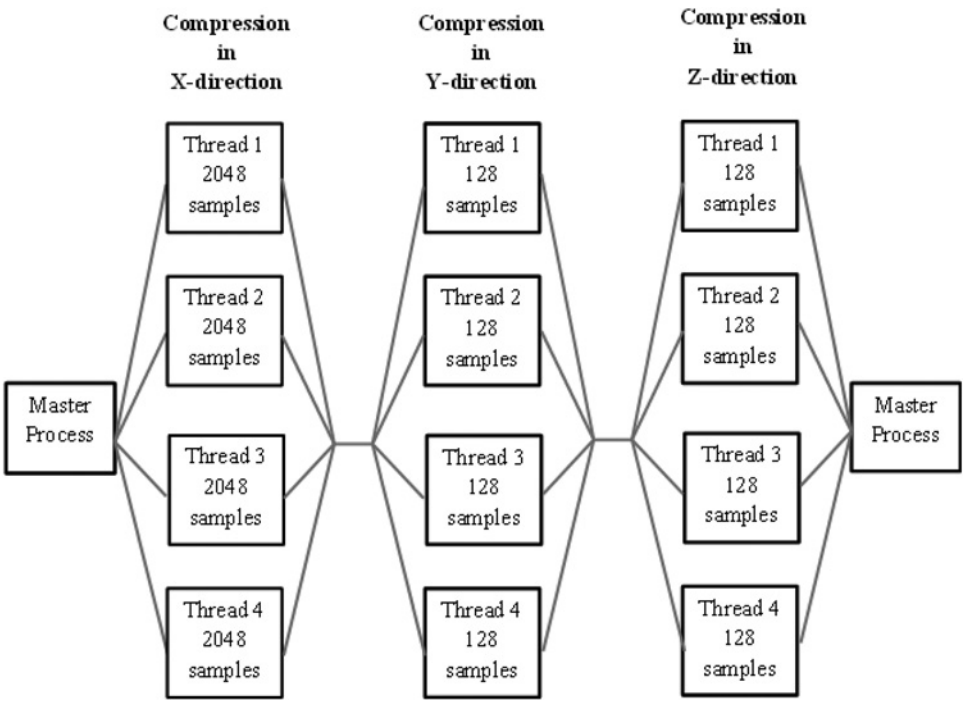

Fig. 7 *Representation of 3D compression algorithm using OpenMP*

OpenMP allows creating parallel programs with less effort and planning with synchronising a particular number of threads.

Fig. 7 shows how the threads are spawned once the program hits the parallel region and then joins back into one thread after completing the parallel operation. When the master thread enters the parallel region, the scheduler spawns a pool of threads which is equal to two times the number of cores in each processor. Since Zynq uses a dual-core ARM processor, there will be four threads (two threads per each core) available to distribute the work load for accelerating the compression and reconstruction processes. The concept is very similar to the *pthreads* in C [37]. We obtain higher efficiency with two threads per core because, when a thread is waiting for an I/O operation to complete, a waiting thread will be scheduled. Thus, the processor idle time is minimised. As shown in Fig. 7, during the decomposition in x-direction, four A-scans will be processed at a time by using four threads, and the iteration continues until all the A-scans are decomposed and compressed. Finally, the threads join together to form the master thread. After this, the master thread will be again split into four threads for executing the decomposition in y-direction, where four 128-sample lines will be processed by using the four threads, until all the lines in y-direction are compressed. Following this, the decomposition is performed in z-direction in a similar way.

For multiprocessing, the threads are managed using the dynamic scheduling paradigm available in OpenMP. This allows the scheduler to dynamically allocate work to the two processor cores at run time, based on their current load. In dynamic scheduling, the processor with lower workload will be allocated with the next set of data to be processed. By this arrangement, both the processor cores are made busy at any given point of time until all the tasks are completed. This ensures parallelism at its maximum, and provides higher computational efficiency. The section of C-program where the compression function is called with enabled parallel processing is shown in (Fig. 8). Note that the data chunk size for the dynamic scheduling is fixed as 4, due to the availability of four threads spanned over two processor cores.

The computational time for software design of the ultrasonic 3D compression algorithm implemented on Zynq dual-core ARM processor is presented in Table 5. From Table 5, it can be seen that, the software design on Zynq ARM processor requires only 1 min for compressing 33 MB of ultrasonic data into 0.42 MB, which indicates that this implementation is highly suitable for real-time ultrasonic imaging applications.

4 Parallel processing using OpenCL

OpenCL provides a programming platform which allows parallel computation of multiple tasks. Unlike traditional C-programs, which are sequential in nature, OpenCL utilises the possibilities of parallelism in hardware circuits [38, 39]. An OpenCL compute device generally consists of many processing elements (PEs). The kernels will be distributed on these PEs by OpenCL application programming interface. The number of tasks which can be executed simultaneously and the number of PEs decide the number of parallel operations, which in turn determine the computational load. In this study, the OpenCL program for ultrasonic 3D data compression algorithm is executed on GeForce GT 750M Nvidia GPU running at 967 MHz, and Intel (R) Core™ i7-4500U CPU running at 1.8 GHz.

```
for (z = 1 ; z < MAX_Z_SIZE; z++)
{# pragma omp parallel //Parallel region starts at this point.
   {# pragma omp for schedule(dynamic, 4)  // Dynamically allocate
                                           // thread chunks of 4.
     for (y = 1; y < MAX_Y_SIZE; y++)
      {compress (signal, signalBuffer, y, z); //2048 samples
      }
   }
}
```

Fig. 8 *Section of C-program where the compression function is called with enabled parallel processing*

In case of 3D compression, a combination of Db10 lowpass and highpass filter is used as the kernel. All the coefficients and input samples are represented as floating point numbers. In the first stage of decomposition, there are 2048 samples within each A-scan that can be processed in parallel. However, due to the limited hardware resources within GPU, only 1024 work items were being identified within the OpenCL compute device which can execute in parallel. These 1024 work items constitute a work group. Nevertheless, since there is a need for sub-sampling, only 1024 alternate samples of the A-scan needs to be processed through the filter as shown in Fig. 9*a*, where each PE performs the computation for a single work item. Thus, each A-scan is processed one at a time (1024 alternate samples within the A-scan are processed in parallel) until all the A-scans ($128 \times 128 = 16{,}384$) are processed through the first stage of decomposition.

In the second stage of decomposition, only 512 alternate samples out of each set of 1024 samples (coming out of stage 1) needs to be processed. Since there are 1024 PEs available, two sets of 1024 samples can be processed simultaneously as shown in Fig. 9*b*. This further improves the speed of computation. Similarly, for the third stage of decomposition, four sets of 512 samples (coming out of stage 2) are processed in parallel. In the fourth stage, eight sets of 256 samples are processed in parallel.

In a similar way, the decomposition in spatial (y and z) directions is also parallelised. In the first stage, 8 sets of 128 samples are processed in parallel, and in the second stage, 16 sets of 64 samples are processed in parallel.

The high level of parallelism enables OpenCL-based implementation to execute the 3D compression algorithm at an extremely high rate compared with the traditional C-based implementation. However, the GPU performance will be slowed down due to the multiple memory transfers required during the execution. Our study demonstrates that even after this bottleneck, GPU outperforms CPU by a factor of two. Table 6 shows the computational time for GPU and CPU implementations for each stage of decomposition.

Table 6 indicates that GPU outperforms the CPU implementation due to the parallel operations, even though the CPU clock is twice faster than GPU clock. Furthermore, as the number of samples gets reduced as we go from stage 1 to 4, the performance factor slightly decreases. However, this performance reduction is minimised due to the fact that multiple input sets are processed in

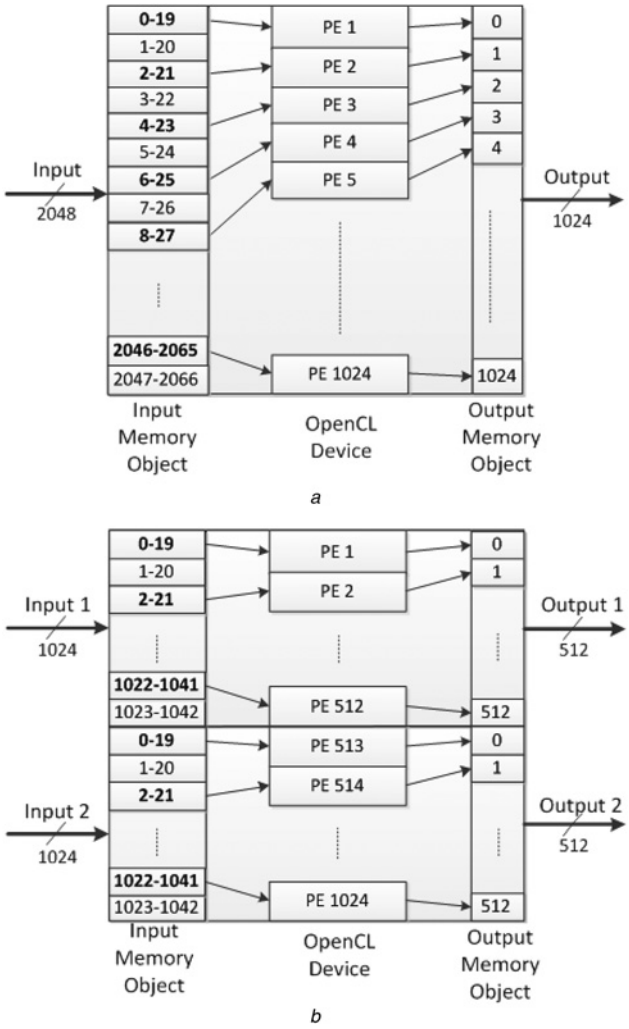

Fig. 9 *OpenCL device structure for*

a First-stage decomposition in x-direction
b Second-stage decomposition in x-direction

Table 5 Computational performance for software design of ultrasonic 3D compression

	Computational time (s) (Zynq ARM Dual Core at 666 MHz)
X – compression	64
Y – compression	1.1
Z – compression	0.25
Overall 3D compression	65.35

Table 6 Computational time for 3D compression implemented on GPU and CPU

Decomposition stage	Computational time on Nvidia GT 750M GPU at 967 MHz (s)	Computational time on Intel i7-4500U CPU at 1.8 GHz (s)	Ratio of CPU time to GPU time
1	0.346	0.564	1.63
2	0.266	0.383	1.44
3	0.135	0.188	1.39
4	0.074	0.099	1.33
Overall	0.821	1.234	1.5

IET Circuits Devices Syst., 2016, Vol. 10, Iss. 1, pp. 54–61

parallel in the stages 2, 3 and 4, apart from the parallel operations of individual samples within each set.

5 Conclusion

Advancement in ultrasonic medical and NDE imaging applications demand huge amount of volumetric data acquisition and real-time processing. Moreover, it has been a common practice to transmit the collected data from remote locations to distant places for expert analysis. Ultrasonic volumetric 3D data compression helps to reduce the storage significantly, and ensures rapid data transmission. The capabilities of recently developed reconfigurable platforms can be efficiently utilised to achieve the performance requirements of real-time ultrasonic imaging applications. This study demonstrates three efficient implementations of ultrasonic 3D data compression algorithms (hardware design and software design on Xilinx Zynq-7020 all programmable SoC platform which embeds an ARM processor unit, and OpenCL design on Nvidia GT 750M GPU). Hardware design compresses the volumetric image of 33 MB into 0.42 MB in 84 ms, whereas the software design completes the compression in about 1 min, making both designs highly suitable for real-time ultrasonic imaging applications. OpenCL implementation provides similar flexibility as that of software design at a higher computational load, requiring around 1 s to complete compression operation, which makes it more attractive, since it is adaptable to various platforms. Furthermore, all three implementations provide a compression ratio of 98.7% with high-quality signal reconstruction.

6 References

1 Kim, I., Kim, H., Griggio, F., *et al.*: 'CMOS ultrasound transceiver chip for high-resolution ultrasonic imaging systems', *IEEE Trans. Biomed. Circuits Syst.*, 2009, **3**, (5), pp. 293–303
2 Kim, G.D., 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
3 Jensen, J.A., Holm, O., Jerisen, L.J., *et al.*: 'Ultrasound research scanner for real-time synthetic aperture data acquisition', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2005, **52**, (5), pp. 881–891
4 Sampson, R., Yang, M., Wei, S., Chakrabarti, C., Wenisch, T.F.: 'Sonic Millip3De: an architecture for handheld 3D ultrasound', *IEEE Micro*, 2014, **34**, (3), pp. 100–108
5 Ahmad, A., Amira, A.: 'Efficient reconfigurable architectures for 3D medical image compression' IEEE Int. Conf. on Field-Programmable Technology, December 2009, pp. 472–474
6 Saniie, J., Oruklu, E.: 'Introduction to special issue on novel embedded systems for ultrasonic imaging and signal processing', *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 2012, **59**, (7), pp. 1329–1332 (invited paper)
7 Licciardo, G.D., Albanese, L.F.: 'Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms', *IET Image Process.*, 2012, **6**, (4), pp. 301–308
8 Meyer-Baese, U., Botella, G., Mookherjee, S., Castillo, E., García, A.: 'Energy optimization of application-specific instruction-set processors by using hardware accelerators in semicustom ICs technology', *Microprocess. Microsyst.*, 2012, **36**, (2), pp. 127–137
9 Xu, M., Yao, H., Huan, X.: 'Performance test of dual-core processor system based on NIOS II'. IEEE Symp. on Electrical & Electronics Engineering (EEESYM), June 2012, pp. 82–85
10 González, D., Botella, G., García, C., Prieto, M., Tirado, F.: 'Acceleration of block-matching algorithms using a custom instruction-based paradigm on a Nios II microprocessor', *EURASIP J. Adv. Signal Process.*, 2013, **1**, pp. 1–20
11 González, D., Botella, G., Meyer-Baese, U., *et al.*: 'A low cost matching motion estimation sensor based on the NIOS II microprocessor', *Sensors*, 2012, **12**, (10), pp. 13126–13149
12 Lili, L., Wanxia, Y., Liqiang, W.: 'The design of dinuclear sewage processor based on the Nios II'. Fifth Int. Conf. on Intelligent Networks and Intelligent Systems (ICINIS), November 2012, pp. 49–52
13 Enfedaque, P., Auli-Llinas, F., Moure, J.C.: 'Implementation of the DWT in a GPU through a register-based strategy', *IEEE Trans. Parallel Distrib. Syst.*, 2014, (99), pp. 1045–9219
14 Darji, A., Arun, R., Merchant, S.N., Chandorkar, A.: 'Multiplier-less pipeline architecture for lifting-based two-dimensional discrete wavelet transform', *IET Comput. Digit. Tech.*, 2015, **9**, (2), pp. 113–123
15 Madanayake, A., Cintra, R.J., Dimitrov, V., *et al.*: 'Low-power VLSI architectures for DCT/DWT: precision vs approximation for HD video, biomedical, and smart antenna applications', *IEEE Circuits Syst. Mag.*, 2015, **15**, (1), pp. 25–47
16 Govindan, P., Saniie, J.: 'Processing algorithms for three-dimensional data compression of ultrasonic radio frequency signals', *IET Signal Process.*, 2015, **9**, (3), pp. 267–276
17 Dobai, R., Sekanina, L.: 'Towards evolvable systems based on the Xilinx Zynq platform'. IEEE Int. Conf. on Evolvable Systems (ICES), April 2013, pp. 89–95
18 Lin, Z., Chow, P.: 'ZCluster: a Zynq-based Hadoop cluster'. IEEE Int. Conf. on Field-Programmable Technology (FPT), December 2013, pp. 450–453
19 Russell, M., Fischaber, S.: 'OpenCV based road sign recognition on Zynq'. IEEE Int. Conf. on Industrial Informatics (INDIN), July 2013, pp. 596–601
20 Song, C., Li, Y., Guo, J., Lei, J.: 'Block-based two-dimensional wavelet transform running on graphics processing unit', *IET Comput. Digit. Tech.*, 2014, **8**, (5), pp. 229–236
21 Garcia, C., Botella, G., Ayuso, F., Prieto, M., Tirado, F.: 'Multi-GPU based on multicriteria optimization for motion estimation system', *EURASIP J. Adv. Signal Process.*, 2013, **1**, pp. 1–12
22 Amaro, J., Falcao, G., Yiu, B.Y.S., Yu, A.C.H.: 'Portable parallel kernels for high-speed beamforming in synthetic aperture ultrasound imaging'. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), May 2013, pp. 2688–2692
23 Wang, G., Xiong, Y., Yun, J., Cavallaro, J.R.: 'Accelerating computer vision algorithms using OpenCL framework on the mobile GPU – a case study'. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), May 2013, pp. 2629–2633
24 Fermin, A., Guillermo, B., Garcia, C., Prieto, M., Francisco, T.: 'GPU-based acceleration of bio-inspired motion estimation model', *Concurrency Comput. Pract. Exp.*, 2013, **25**, (8), pp. 1037–1056
25 Saniie, J.: 'Ultrasonic signal processing: system identification and parameter estimation of reverberant and inhomogeneous targets'. PhD thesis, Department of Electrical Engineering, Purdue University, West Lafayette, IN, 1981
26 Oruklu, E., Saniie, J.: 'Dynamically reconfigurable architecture design for ultrasonic imaging', *IEEE Trans. Instrum. Meas.*, 2009, **58**, (8), pp. 2856–2866
27 Desmouliers, C., Oruklu, E., Saniie, J.: 'Discrete wavelet transform realisation using run-time reconfiguration of field programmable gate array (FPGA)s', *IET Circuits Devices Syst.*, 2011, **5**, (4), pp. 321–328
28 Zhao, L., Jia, Y., Xie, Y.: 'Robust transcale decentralised estimation fusion for multisensor systems based on wavelet packet decomposition', *IET Control Theory Appl.*, 2014, **8**, (8), pp. 585–597
29 Djawad, Y.A., Kiely, J., Nibouche, M., Wraith, P., Luxton, R.: 'Robust feature extraction from impedimetric signals using wavelet packet decomposition with application to cytotoxicity testing', *IET Sci. Meas. Technol.*, 2012, **6**, (6), pp. 456–463
30 Govindan, P., Saniie, J.: 'Performance evaluation of 3D compression for ultrasonic nondestructive testing applications'. IEEE Int. Ultrasonics Symp. (IUS), July 2013, pp. 437–440
31 Govindan, P., Saniie, J.: 'Hardware-software co-design of 3D data compression for real-time ultrasonic imaging applications'. IEEE Int. Ultrasonics Symp. (IUS), September 2014, pp. 564–567
32 Proakis, J.G., Manolakis, D.G.: 'Digital signal processing: principles, algorithms and applications' (Pearson, 2011, 4th edn.)
33 Parhi, K.K.: 'VLSI digital signal processing systems: design and implementation' (Wiley, 1999, 2nd edn.)
34 Yeary, M.B., Zhang, W., Trelewicz, J.Q., Zhai, Y., McGuire, B.: 'Theory and implementation of a computationally efficient decimation filter for power-aware embedded systems', *IEEE Trans. Instrum. Meas.*, 2006, **55**, (5), pp. 1839–1849
35 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
36 Hunag, L., Stotzer, E., Yi, H., Chapman, B., Chandrasekaran, S.: 'Parallelizing ultrasound image processing using OpenMP on multicore embedded systems'. IEEE Global High Tech Congress on Electronics, November 2012, pp. 131–138
37 'IEEE Standards Interpretations for IEEE Std 1003.1c™-1995 IEEE Standard for Information Technology–Portable Operating System Interface (POSIX®) – System Application Program Interface (API) Amendment 2: Threads Extension (C Language)'. Available at http://standards.ieee.org/findstds/interps/1003-1c-95_int/, accessed January 2015
38 'The Open Standard for Parallel Programming of Heterogeneous Systems'. Available at https://www.khronos.org/opencl/, accessed January 2015
39 Jaaskelainen, P.O., De La Lama, C.S., Huerta, P., Takala, J.H.: 'OpenCL-based design methodology for application-specific processors'. IEEE Int. Conf. on Embedded Computer Systems (SAMOS), July 2010, pp. 223–230