Microprocessors and Microsystems 64 (2019) 23–33

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

# A high performance real-time Interferometry Sensor System Architecture

Tassadaq Hussain[a,b,*], Saqib Amin[a], Usman Zabit[a], Olivier D. Bernal[c], Thierry Bosch[c]

[a] Riphah International University Islamabad, Pakistan
[b] UCERD, Islamabad, Pakistan
[c] LAAS-CNRS, Université de Toulouse, CNRS, INPT, Toulouse, France

ARTICLE INFO

*Article history:*
Received 28 September 2017
Revised 9 July 2018
Accepted 18 October 2018
Available online 19 October 2018

ABSTRACT

Optical feedback or self-mixing interferometry technique has been widely used for sensing vibration, displacement, velocity, distance and flow applications. Such applications require an accurate and consistent output for real-time target measurements. Furthermore, array based sensing, as opposed to point sensing, is being increasingly pursued which requires multiple-input, parallel computing platform. In this article, we have proposed and developed a real-time interferometric sensor applications based multi-core system architecture and programming toolkit. To show that our system is useful in a variety of situations, we applied three algorithms of varying complexity and accuracy for displacement and vibration measurement. In order to prove the efficiency of proposed sensor processing system, we compared it performance and power consumption with a state of the art NXP LPCX54102 sensor processing and motion system architecture. When compared with the baseline multi-core system, the results show that our system improves the system performance upto 7.55 times, draws 15.6% less dynamic power and consumes 8.9 times less energy.

## 1. Introduction

To improve the sensor system performance, the system architect emphasizes technology unification and rational function. In fact, it is not a particular technology or particular material that make up smart sensor system architecture, but the synergistic interfacing among the essential components that make up the sensory system. Seamless, useful interfacing challenges the sensor system designer to understand fundamental scientific and design principles derived from material science, electronics, optics, and informatics. These branches of science, engineering, and information give knowledge about the design and application of practical sensor architecture.

Because of the increase and improvement in the digital electronics, many manufacturers [1,2] provide high performance single board computer architectures for a large range of applications. Some manufacturers [3,4] supply development kits for sensor processing and motion solution applications. These high performance single board computers use data acquisition system which collects the data and provides it to processing core for computation. Similarly, with the increase in the performance and density of Field Programmable Gate Arrays (FPGA)s [5,6] allows the sensors industry to develop high-precision and high-performance systems, which can execute complex real-time sensor applications on a single chip. However, these system architectures do not have high speed and low-power analog front end interface that gathers real-time data from the sensor. These architectures also lack an efficient memory system to hold, update and reuse sensors information, high performance processing unit that executes multiple computational tasks in parallel using hardware/software co-design approach, and an easy to use programming environment that helps sensor algorithm programmer.

In this article, we develop an easy to program, low-power and high-performance Interferometry Sensor System Architecture (ISSA). ISSA takes complex high-speed data from interferometry sensors interface or saved in the memory, handles it in an on-chip memory system and processes them using dedicated hardware accelerators or multi-RISC (Reduced Instruction Set Computer) processor system. The ISSA provides single-board hardware and software solution for sensor applications, by accessing high-speed data using Front-end Sensor Interface, placing it to local memory and processing it using heterogeneous multi-core processing system. To hide the hardware complexity the ISSA provides sensor applications programming toolkit which provides data transfer, memory management, and processing function calls. In order to val-

* Corresponding author at: UCERD, Islamabad, Pakistan.
*E-mail address:* tassadaq@ucerd.com (T. Hussain).

https://doi.org/10.1016/j.micpro.2018.10.006
0141-9331/© 2018 Published by Elsevier B.V.

idate the ISSA three diverse sensor algorithms (CSU, IPUM, and TFSP) are targeted. CSU algorithm is the least precise algorithm among the three algorithms. However, due to the simplest processing, CSU is most efficient in terms of resources, speed, and power consumption. IPUM is a most precise algorithm, but more precision is achieved at cost of more resources, power and time utilization. TFSP is in-between CSU and IPUM in terms of precision, resource utilization, time and power consumption. Salient contributions of the proposed ISSA are:

- Enables hardware/software co-design and parallel processing approach, by supporting multiple general purpose RISC and application specific hardware accelerator cores.
- Gives high-performance and real-time application support by using a memory system which manages computer intensive and time critical data and their complex transfers.
- Provides hardware reconfigurability by using application specific accelerators which allow sensor system architect to port more features in hardware without changing the system architecture.
- Consumes low power by utilizing a front-end interface that handles multiple sensors and their 1D/2D data transfers.
- Provides energy efficacy and high speed by using an on-chip scheduler which handles multiple sensors and processing cores.
- Offers programming toolkit which helps sensor application programmers to write their code easily and efficiently.
- When compared to a baseline sensor system, the ISSA improves the application performance up to 7.5 times, draws 15.6% less dynamic power and consumes 8.9 times less energy.

The remainder of this paper is prepared as follows: Sections 2 and 3 discuss related work and generic self-mixing interferometry sensor system and algorithms respectively. Section 4 describes the internal architecture and working principle of ISSA. Section 5 describes ISSA and baseline system implementation and evaluation on an FPGA-based prototype environment. Section 6 presents results. Finally, Section 7 provides the conclusions.

## 2. Related work

Interferometric sensor applications have expanded considerably over the last few years Although many of these sensors are expectedly unitary providing point sensing but now many multi channel and array based sensing configurations are also maturing due to availability of dense monolithic laser diodes [7]. It is expected that array based sensing would lead to grid based sensing for imaging of specimen under observation. Therefore, the need of high performance, scalable and multi-channel architectures supporting parallel processing can not be understated. Details of previously published multi-channel as well as unitary sensing systems and architectures are presented below.

Lim et al. designed a SM flow sensor with parallel readout [7] by using a monolithic LD array composed of 12 lasers. The proposed parallel sensing enabled high-resolution full-field imaging systems employing electronic scanning with faster acquisition rates than mechanical scanning systems. The scheme can be expanded to envisage massively parallel Doppler imaging systems based on 2D laser arrays. However, it may be noted that in spite of availability of 12 simultaneous sensor channels, the sensing system only processed one channel at a time by employing electronic switching. Such a hindrance would be completely removed if a sensing system architecture of the kind of ISSA is used as it would enable not only the acquisition of multiple sensing channels but also their parallel processing.

Esteban et al. [8] proposed multi-channel phase meters on an FPGA device for IR-interferometers . Several optimizations for multichannel systems are described and tested on the existing prototype systems.

Wang [9] designed a 4-channel phase meter prototype, which can perform the currently-used commercial phase measurement solution in the laboratory. The system is designed by using System-on-Programmable-Chip (SOPC) and tested on Altera DE2-115 FPGA board.

Strube et al. [10] proposed an FPGA-based multi-axial interferometer for simultaneous tilt and distance measurement. The system allows multiple measuring modes, e.g. intermittent contact mode or frequency tracking mode.

Vera-Salas et al. [11] designed a smart sensor for online displacement measurements using heterodyne interferometry. The design, capable of large displacement measurements, provides high resolution and accuracy. Smart processor is implemented on Spartan 3E XC3S1600E FPGA operating at 125 MHz.

Isleif et al. [12] developed a high-speed multiplexed heterodyne interferometer for measurement of multiple targets separated by only a few centimeters. An FPGA splits the acquired photodiode signal into independent readout channels which are further processed for the extraction of phase and amplitude.

Merlo et al. [13] proposed a numerical analysis based signal processing approach to reconstruct target displacement. Results shown that accuracy of10's of nano-meter was achieved for displacement reconstruction. However parameters required to perform numerical analysis can only be extracted from week feedback regime signal, which limits the use of this technique for rich variety of SM signals.

Norgia et al. [14] demonstrated a SM flow sensor by measuring the Doppler shift inside fluid due to scattering particles. The prototype used a digital signal processor (DSP) model TMS320F28044 from Texas Instruments, with 6 MHz sampling frequency. Norgia et al. [15] also developed an SM sensor for distance and vibration measurement. It uses FFT routine implemented on an FPGA to estimate the distance.

Cavedo et al. [16] developed a low power and reduced size optical sensor for distance measurement to characterize steel pipes. The prototype has a resolution of 10 µm for a measurement range of 10 cm. Microcontroller STM32F303 was used along with a 32-bit ARM Cortex-M4 core operating at 72 MHz.

Magnani et al. [17] presented a real-time SM optical interferometer, that can reconstruct speed and direction of the target, without heterodyne detection. The system is implemented on Altera DE0-nano FPGA board. Magnani et al. [18] also proposed a SM vibrometer using a DSP which acquires the SM signal and reconstructs the target vibration. The proposed systems have less hardware architectural and design details.

Thus to summarize, some GPP, DSP and FPGA based sensing system architectures are proposed in the above mentioned related work. These systems have standard hardware architectures, which are programmed by standard general purpose Programming Toolkits. On the other hand, we propose an ISSA structure that gives following features:

- Provides *parallel processing* and *hardware software co-design approach*, by supporting multiple RISC processor instruction set architecture and reconfigurable hardware accelerators that can execute a variety of real-time sensing applications. The *Processing System* performs computation on the input signal in software as well as in hardware. The system uses programmable multi-core ARM Cortex-A9 processors for general purpose applications and SASHA cores are integrated for high-performance applications.
- Enables *reconfigurability*, which allows sensor system architect to port more features in hardware without changing the structure of ISSA.

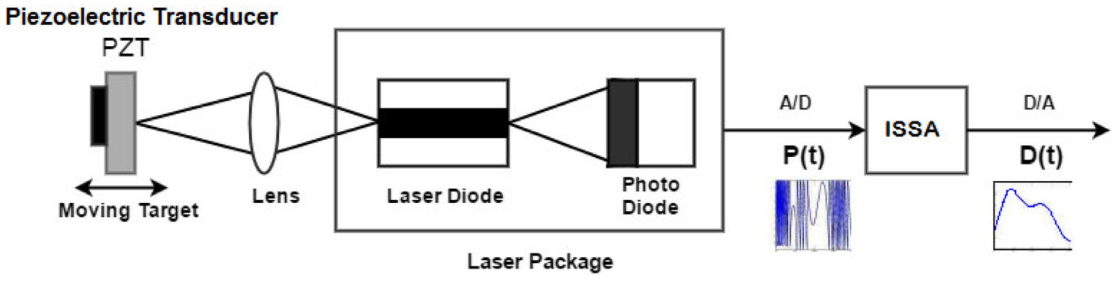


**Fig. 1.** Typical self-mixing displacement sensor setup.

- Supports *high-performance* and *real-time* applications by using a *Memory System* which manages complex data transfers using scratchpad and cache memories.
- Consumes *low power* by integrating an ultra *low-noise* and low-power *Front-end Interface* that arranges data from multiple sensor units.
- Provides *energy efficacy* and high speed by using an on-chip scheduler which handles multiple sensors and processing cores without support of master processor or operating system.
- Proposes a *Programming Toolkit* for sensor algorithms which helps the developer to write an application without needing to know low-level hardware details.

## 3. Self-mixing interferometry sensor system

### 3.1. Introduction to self-mixing interferometry

Self-Mixing (SM) or optical feedback interferometry has been regularly used for metrological measurements during the last two decades [19,20]. As opposed to classical dual-arm interferometers such as Michelson interferometer, SM enables a compact, self-aligned, and low-cost sensor design with minimal optical part count [19,20]. SM effect occurs in a laser when a fraction of the beam is back-scattered by a remote target into the laser cavity to cause interference with the emitted beam, resulting in modifications of optical and spectral properties of the laser [19]. Peculiarly, under SM, the laser simultaneously acts as a light source, a micro-interferometer, as well as a coherent detector.

The characteristics of SM interferometric signal depend on the optical feedback coupling factor $C$ and the LD linewidth enhancement factor $\alpha$ [21]. Typical SM sensing setup is schematized in Fig. 1. An experimentally acquired SM signal (using Sanyo DL7140 laser diode emitting at 785 nanometer) can be seen in Fig. 2.(b) for an arbitrarily moving remote target. Usually, a laser package (containing the laser diode and built-in monitoring photodiode) is used with a focusing lens so that the laser illuminates the remote target surface. The acquired SM signal having variations of output optical power $P(t)$ is then processed to retrieve $D(t)$ as detailed below.

### 3.2. SM displacement sensing

Due to SM, the free running laser wavelength $\lambda_0$ is modified and becomes a function of time $\lambda_f(t)$ varying with $D(t)$. It can be found by solving the excess phase equation [20]:

$$\Phi_0(t) = \Phi_f(t) + C\sin[\Phi_f(t) + a\tan(\alpha)] \quad (1)$$

where $\Phi_f$ and $\Phi_0$ represent two phase signals (subject to feedback and under free running conditions, respectively) written as a function of the wavelengths $\lambda_f(t)$ and $\lambda_0$, respectively:

$$\Phi_f(t) = 4\pi D(t)/\lambda_f(t) \quad (2)$$

$$\Phi_0(t) = 4\pi D(t)/\lambda_0 \quad (3)$$

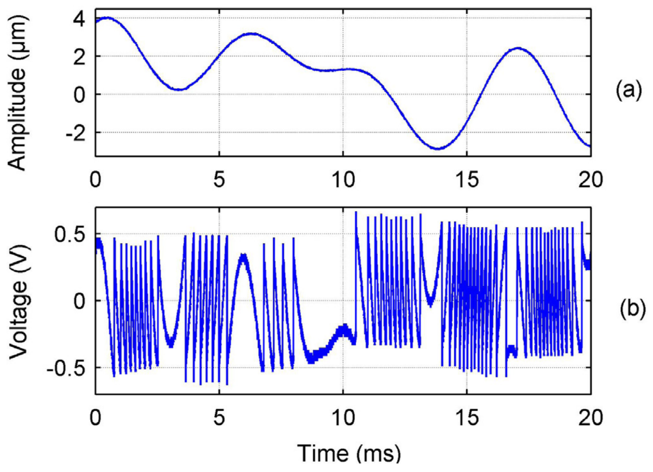


**Fig. 2.** (a) Remote target motion and (b) corresponding experimentally acquired SM signal using Sanyo DL7140 laser diode emitting at 785 nm.

The value of $\Phi_f(t)$ can be extracted from $P(t)$ by using

$$P(t) = P_0[1 + m\cos(\Phi_f(t))] \quad (4)$$

where $P_0$ is the free running LD's power and $m$ a modulation index.

So, by using $\Phi_f(t)$, $\Phi_0(t)$ can first be estimated by using any one of many algorithms [22–28] and then $D(t)$ is obtained as per Eq. (3). These various algorithms differ from one another in terms of the level of detail at which the retrieved $\Phi_f(t)$ is processed by each of them. Thus, varying processing complexity leads to varying measurement accuracy ranging from $\lambda_0/2$ to $\lambda_0/50$ [22–28].

### 3.3. Test algorithms

In this paper, three algorithms/applications are used to retrieve remote target's motion by processing the SM signal. Brief description of each of these is provided in Table 1.

The CSU algorithm is based on a direct laser phase unwrapping approach [24]. The CSU algorithm retrieves the target's displacement by initially detecting the forward and backward fringes present in the SM signal. Then, a staircase signal is achieved by an integration of all detected fringes resulting in an initial resolution of $\lambda_0/2$. Finally, SM signal is appropriately scaled and merged with the staircase signal in order to better retrieve the target motion.

The TFSP algorithm [26] performs a frequency domain analysis of the retrieved phase $\Phi_f(t)$. The application then identifies the frequency components that make up the main modes of the target vibration. The unwanted noise and distortions thus identified in the frequency domain are then filtered allowing retrieval of target vibration with better accuracy.

The IPUM is the most sophisticated method considered in this paper for the estimation of laser phase (see Fig. 3). It allows improved unwrapping of $\Phi_f(t)$ by correcting local phase inversions in

**Table 1**
Algorithms specifications.

| | CSU | TFSP | IPUM |
|---|---|---|---|
| Accuracy | λ/25 | λ/50 | λ/75 |
| Computational blocks | Adder Normalization | FFT IFFT | max/min search 2D optimization |
| Processing Complexity | Light | Medium | Heavy |

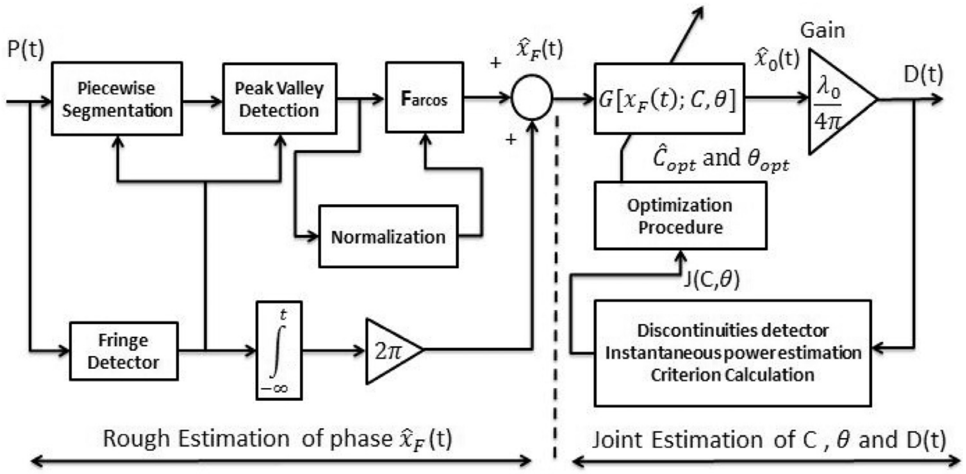


**Fig. 3.** Block diagram of the Improved Phase Unwrapping Method (IPUM) [22].

**Table 2**
Errorcompare of CSU, IPUM and TFSPalgorithms against various optical feedback strength signals.

| Experimental signal no. | Estimated C value | RMS error (nm) | | | Peak error (nm) | | |
|---|---|---|---|---|---|---|---|
| | | CSU | TFSP | IPUM | CSU | TFSP | IPUM |
| 1 | 0.77 | 115 | 82 | 18 | 200.4 | 120.9 | 44 |
| 2 | 1.25 | 88.6 | 16.2 | 11 | 212.5 | 229 | 35 |
| 3 | 1.27 | 76.4 | 18.4 | 16 | 232.3 | 26.1 | 33 |
| 4 | 2.07 | 82.6 | 25.5 | 22 | 265.9 | 32.69 | 53 |
| 5 | 2.83 | 86.4 | 19.3 | 8 | 272.5 | 27.27 | 25 |
| 6 | 4.6 | 102.6 | 15.46 | 6.88 | 284.8 | 22 | 20.8 |
| Average error | | 91.33 | 29.48 | 13.65 | 244.7 | 41.98 | 35.13 |

$P(t)$ through the use of piecewise segmentation. An iterative joint estimation of feedback parameters then allows solution of Eq. (4). In the absence of noise, IPUM has a potential of delivering sub-nanometric accuracy [22].

### 3.3.1. Tolerance of algorithm

In order to compare the error of CSU, IPUM and TFSP algorithms, we have conducted an experiment for various optical feedback strength signals, and their results are shown in Table 2. Table presents peak and RMS values of errors for various experimental signals, it also presents the average tolerance of algorithms calculated from the average of different optical feedback strength signals.

## 4. Interferometry Sensor System Architecture

The Interferometry Sensor System Architecture (ISSA) (shown in Fig. 4) includes the sensor front-end design, memory management unit, processing system and programming toolkit. The section is further divided into six subsections: the *Overview of ISSA*, the *Front-end Interface*, the *Memory System*, the *Scheduler*, the *Processing System* and the *Programming Toolkit*.

### 4.1. Overview of ISSA

ISSA supports multiple sensors and processing cores using the system on chip *Scheduler*. ISSA reserves *Program Memory* for the sensor application in the form of descriptors [29,30], each descriptor holds data transfer, memory and processing information of an application. Depending upon the program memory, the *Front-end Interface* accesses analog data from single or multiple sensors, performs digitization and basic analog operations and places them on the *Memory System*. The *Memory System* uses *Scratchpad, Cache* and *Main* memories for data storage and *Program Memory* for control and data computation information. At run-time, the ISSA *Scheduler* takes single or multiple requests from sensor algorithms and creates a link between the *Front-end Interface* and the *Processing System*. Depending upon the sensor's data requests the *Scheduler* takes single or multiple descriptors from the *Program Memory* and schedules the data movement between the *Memory System* and the *Processing System*. The *Scratchpad* and *Cache* memories manage multiple sensor data and feeds it to the *Processing System*. The *Main Memory* is used to hold sensor data if it does not fit in *Scratchpad* or *Cache*. The *Processing System* uses Sensor Application Specific Hardware Accelerators (SASHA) and multi-core RISC processors to execute the sensing algorithms.

### 4.2. Front-end interface

ISSA is interfaced with multi-sensors using *Front-end Interface* (FEI). The *Front-end Interface* of ISSA consists of signal drivers and multiple analog-to-digital converters (ADC) shown in Fig. 5. The current structure of ISSA supports maximum of 16 paral-

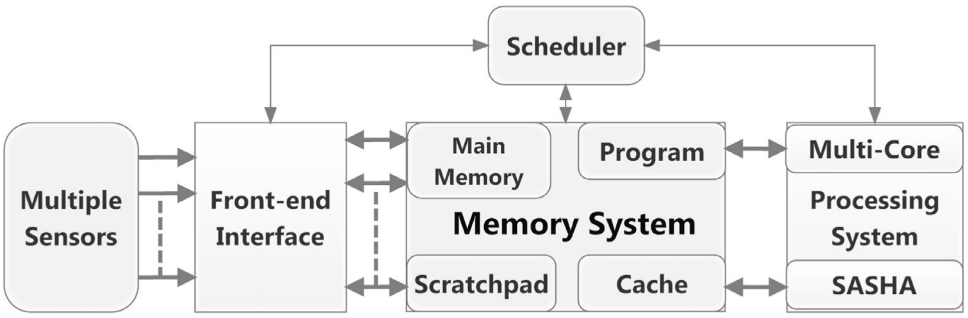
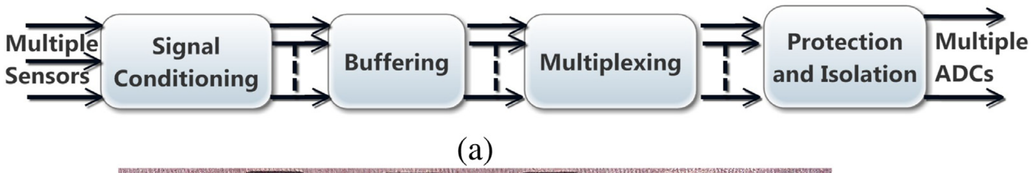


**Fig. 4.** ISSA: Interferometry Sensor System Architecture.



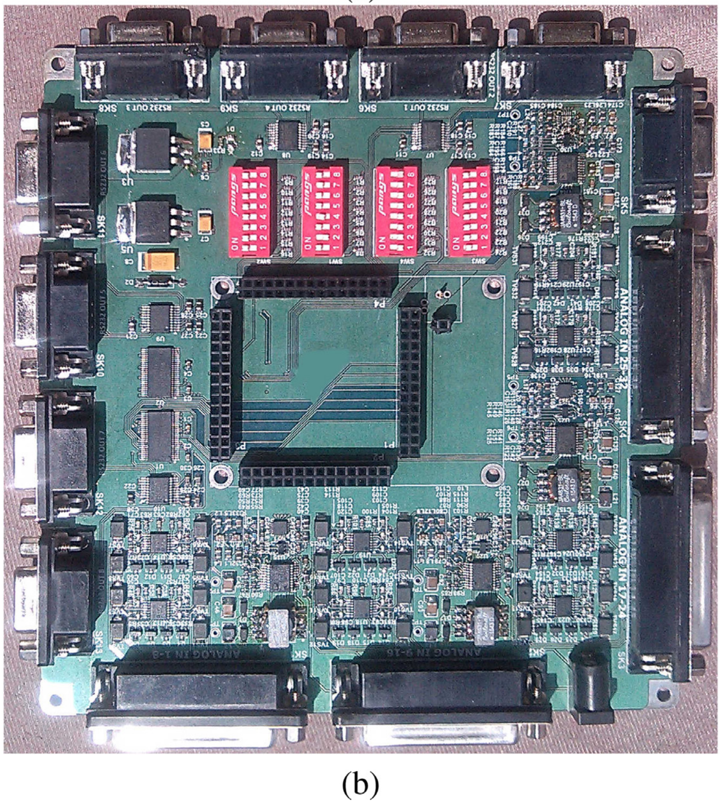


**Fig. 5.** (a) Blockdiagram of ISSA front-end interface (b) Photograph of developed front-end interface card.

lel analog signals. The primary objective of the signal driver is to achieve functions like buffering, amplitude scaling, single-ended-to-differential and differential-to-single-ended conversion, common-mode offset adjustment, and filtering. The *Front-end Interface* receives an electrical signal from multi-sensors and transfer it to signal driver. After performing basic operations, the signal driver moves signal to ADC. The block diagram of single *Front-end Interface* is shown in Fig. 5 (a) and a photograph of the developed *Front-end Interface* printed circuit board is seen in Fig. 5 (b) while salient features are mentioned in Table 3. The *Front-end Interface* takes input signal with frequency up to 6 MHz, minimum input voltage of 2 $\mu$V. The input signal can be single ended, differential or pseudo differential. The *Signal Conditioning* block performs electro magmatic interference and transient voltage suppression.

**Table 3**
Salient characteristics of *Front-end Interface* after bench testing.

| | |
|---|---|
| Maximum bandwidth | 6 MHz maximum (while tested at 500 KHz). |
| Slew rate | 2 V/$\mu$s (extended to 8 V/$\mu$s) |
| Gain | Adjustable gain up-to max 1000 |
| Maximum analog input range | $\pm$20 V (clipping network for above or below $\pm$20 V range) |
| Input offset | 0.4 mV |
| Common mode Rejection (CMR) | 110 dB |
| Input impedance for differential mode operation | 0.4 M ohm |
| Input impedance for common mode operation | 11 G ohm |

The *Front-end Interface* uses two gain stages for signal conditioning to make week input signal readable for the ADC. The input voltage level for the *Front-end Interface* ADC is ±10 V, for this reason the maximum output voltage is set to this range. The *Buffering* state uses buffer for reduction of loading effect and impedance matching. A low pass filter is also present in the buffer stage. The *Multiplexing* stage uses two 8x1 multiplexers with separate enable controls. Later *Protection and Isolation* stage uses voltage level protection circuit, for the protection of ADC. Isolation circuit provides significant isolation between ADC and the input signal. The *Front-end Interface* uses 16 parallel successive approximation based ADCs (ADAS3023). Each ADC has maximum rate of 1MSPS with 16-bit resolution.

### 4.3. Memory system

The ISSA uses four types of memories: the *Program Memory*, the *Cache*, the *Scratchpad Memory* and the *Main Memory*.

#### 4.3.1. Program memory

The ISSA uses descriptors [29–31] to manage data transfer requests of multi-sensors. These descriptors reduce the overhead of run-time request generation, memory, and data transfer management. The set of parameters for a descriptor block includes *Sensor Address, Memory Address, Priority* and *Size*. The *Sensor Address* parameter specifies the sensor unit. In case of multiple sensors, each sensor will be allocated a different *Sensor Address*. The *Memory Address* specifies the memory location to place the sensor data. The *Priority*, when combined with other priorities, determines the order in which sensor data is entitled to be processed. The *Size* defines the number of data samples to be transferred.

#### 4.3.2. Cache

The *Cache* is used to dynamically store a subset of the frequently used data. In our current design *Cache* memory is used for the non-real time applications. Thus, the timing of a read/write operation depends on the relationship between its effective address and the valid addresses of earlier operations. The *Cache* is managed and used by the *Multi-RISC* cores of the *Processing System* (see Section 4.5). When the processing core makes a memory access (read data, or write data), the *Memory System* first checks if the cache reads the data. The *Cache* monitors the access requests to check if it holds the requested data. If it does, a cache hit occurs, and the cache bypasses the *Memory System* and provides the data to the processing core. Otherwise, a cache miss occurs, and the *Memory System* contains the data and its surrounding elements from the *Main Memory*, the processing core receives the data and the cache stores the whole data elements as a cache line.

#### 4.3.3. Scratchpad memory

A programmable *Scratchpad Memory* [29] architecture is introduced in the system. The structure of *Scratchpad Memory* is programmed according to the application access pattern. The *Scratchpad Memory* structure is used to reduce the data access latency by accessing noncontiguous data set from multiple sensors, place them in contiguous format on *Scratchpad Memory* and provide single cycle data access to the processing core.

#### 4.3.4. Main memory

The *Main Memory* is the slowest type of memory having DRAMs. The *Main Memory* has high data density and read/write latency compared to the *Scratchpad and Cache* memories. Therefore, it is used to execute non-real time applications having big data volume.

**Table 4**
C/C++ function calls toprogram Interferometry Sensor System Architecture.

| API Function | Description |
|---|---|
| `FEI ( MULTI-SENSOR, MEMORY_SYSTEM, Priority)` | *Sensor Front-end Interface Data Access* `MULTI-SENSOR` indicates a *sensor* unit `MEMORY_SYSTEM` indicates *memory* to write data and *Priority* defines the order of execution |
| `ALGORITHM( Read_MEMORY, Write_MEMORY, Priority, Processor_Core, Parameters)` | *Processing Application* `MEMORY` indicates the read and write memories, *Priority* defines the order of execution, and `Processing Core` indicates the type of processor core and its parameters |

### 4.4. Scheduler

The ISSA *Scheduler* handles multi-sensors and multiple processing cores. The *Scheduler* accesses data of the multi-sensors using the programmed descriptor memory, which reduces the input/output interfaces and multiple processor communication time. At run-time, the ISSA *Scheduler* receives multiple data read/write requests from the Multi-core *Processing System*, selects a core and creates a sensor link depending upon its priority level. The Scheduler also performs run-time automated scheduling [32]. The automated scheduling executes the algorithm task based on run-time available resources. For example, for an ISSA having dual-core processing system the automatic scheduling execute the two sensor algorithms in parallel and automatically pipeline the others.

### 4.5. Processing system

The *Processing System* supports multiple Sensor Application Specific Hardware Accelerator (*SASHA*) and *Multi-RISC System*. SASHAs are used in the design to execute applications in hardware, as they consume low hardware resources, draw less power and give high-performance computation. The SASHA grabs sensor's data from *Scratchpad Memory* and executes specific applications. The SASHA are designed in hardware using hardware description language (HDL) or C/C++ high level synthesis (HLS) languages. *Multi-Core System* uses multiple RISC instruction set processor architecture, which provide programmability, flexibility and software data processing. The *Multi-Core System* executes multiple applications in software by using C/C++ programming. The ISSA scheduler can integrate multiple heterogeneous processing cores. The ISSA can execute application algorithm using two mode of processing: the ISSA *Multi-Core System* (I-MCS) and the ISSA multiple *SASHA* (I-SASHA). The *I-MCS* executes the application algorithms in software using multiple RISC cores, whereas the *I-SASHA* processes algorithms on application specific hardware.

### 4.6. Programming toolkit

The sensing algorithms have different design constraints and require a user friendly programming interface to write the application for multiple sensors and processing cores. ISSA *Programming Toolkit,* intends to eliminate the stress of manually fixing sensor's data transfers and satisfies the performance demands of algorithms. When using ISSA Programming Toolkit, the programmer does not need to take care of the related hardware constraints. ISSA supports function calls to transfer data between the *Front-end Interface* and the *Memory System*, performs data management and executes different applications on the *Processing System*. Table 4 shows function calls to program the current ISSA. The *Processing Core* interacts with ISSA through controls, status signals and data

```
#define 1D 1024
#define 2D 32
#define points 1024

/* Part I */
/* 1K Byte Scratchpad Memory */
Scratchpad_Memory Read_mem0[1D];
Scratchpad_Memory Write_mem0[1D];
/* 1024 x 16 K Byte Scratchpad Memory */
Scratchpad_Memory Read_mem1[1D][2D];
Scratchpad_Memory Write_mem1[1D][2D];

/* Part II */
/* Read Data from Multi-Sensors */
FEI(Read_mem0, sensor0, Priority);
FEI(Read_mem1, sensor1, Priority);

/* Part III */
/* Algorithms Computation */
FIR(Read_mem0, Write_mem0, Priority, SASHA, Points);
FFT(Read_mem1, Write_mem1, Priority, Multi-RISC, Points);
```

**Fig. 6.** FIR and FFT examples using ISSA programming toolkit.

registerers. ISSA provides a wrapper library that offers function calls to describe the *Multi-Sensors* data transactions, *Memory System* for data management and *Processing System* for the computation.

The programmer only needs to identify the appropriate function call available in the library, and the ISSA automatically manages/handles the sensor data to the *Memory System* of the *Processor System*. The proposed ISSA provides C and C++ language support. Fig. 6 shows an example of ISSA *Programming Toolkit. Part I* defines a 1D and 2D *Scratchpad* read/write memories of 1024 and 1024x32 Bytes respectively. The program structure is used to initialize a 1D and 2D *Scratchpad Memory*. The value (1D) 1024 describes the size of a row (`1D Buffer Width`) of the *Read_mem0* and *Write_mem0 Scratchpad* memories. 1024 $\times$ 32, 2D-dimensional *Read_mem1* and *Write_mem1 Scratchpad* memories are defined by assigning [1024][32] ([1D][2D]) parameters. The current ISSA *Scratchpad Memory* architecture is composed of single or multiple BRAM blocks. *Part II* initializes the *Multi-sensors* data transfer information using *Front-end Interface. Part III* indicates al-

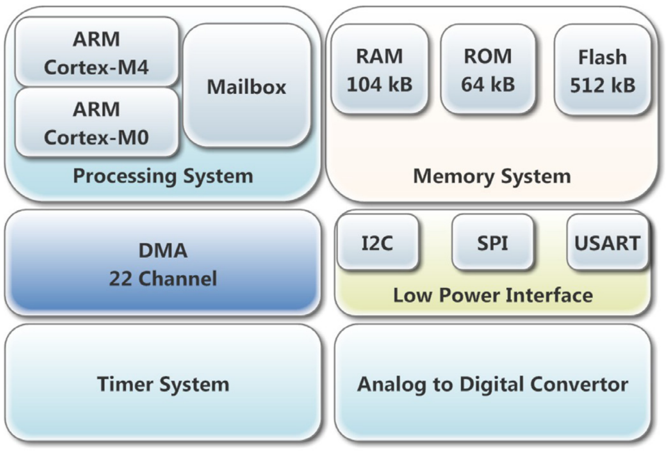


**Fig. 8.** NXP LPC54102:baseline interferometry sensor system.

gorithm functions (FIR and FFT) which perform computation on sensor's data.

## 5. Prototype architecture

In this section, we describe Baseline System and Zynq-based ISSA. The Vivado Design Suite is used to design the ISSA. A state-of-the-art, high-performance, low-power, 28 nm ZYNQ 7000 series FPGA based device is used. Thus, two modes of ISSA, i.e. I-MCS and I-SASHA both are implemented on ZYNQ 7000. For comparison, an NXP OM13078 single board computer having LPC54102 processing system is chosen as the Baseline Interferometry Sensor System. An image of the SMI laboratory setup is shown in Fig. 7. The section is further categorized into two subsections which are detailed below.

### 5.1. Baseline interferometry sensor multi-core system

The Baseline Interferometry Sensor Multi-Core System (*B-MCS*) [33] (shown in Fig. 8) comprises a dual core ARM processor, (ARM Cortex-M4 and Cortex-M0+ coprocessor). Mailbox is used for inter-process communication and task sharing for a same application. For external peripheral communications and interfacing sys-

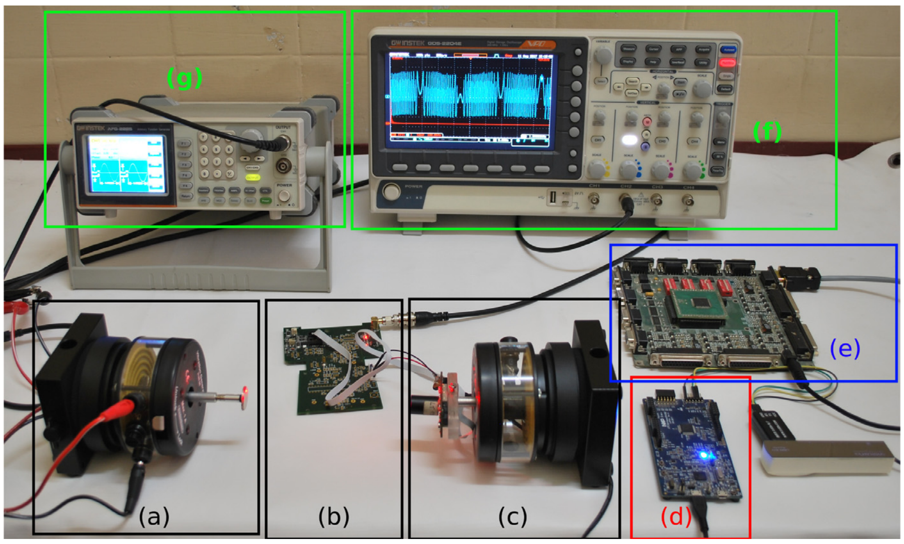


**Fig. 7.** SMI laboratory setup: (a) PASCO SF-9324mechanical shaker used as moving target (b) DL7140 laser diode based sensor electronics circuitry (c) DL7140 based laser sensor mounted on mechanical shaker (d) NXP: baseline interferometry sensor multi-core system (e) Zynq based ISSA (f) GWINSTEK GDS-2204E oscilloscope to observe generated SM Signal (g) GWINSTEK AFG-2225function generator used to excite mechanical shaker at a frequency of 80 Hz .

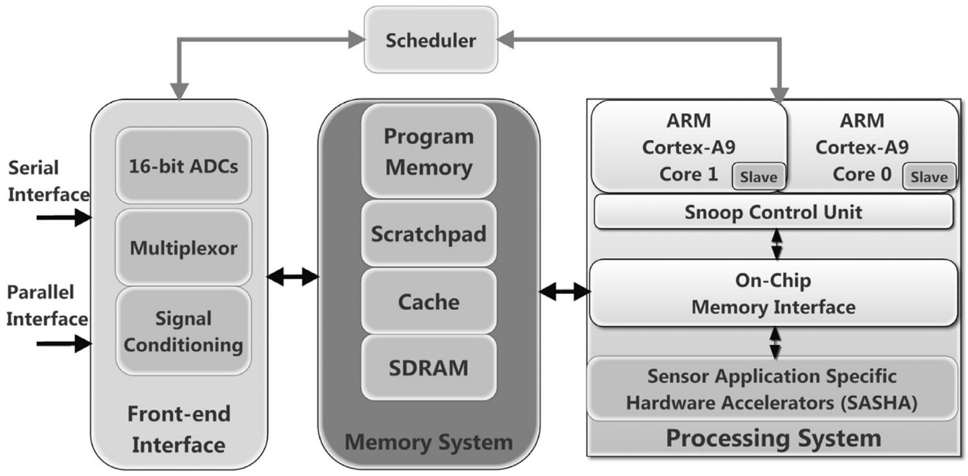


**Fig. 9.** Zynq based ISSA.

tem uses four USARTs, two serial peripheral interfaces (SPIs), three Fast-mode Plus I2C-bus interfaces with high-speed slave mode, and one 12-bit 5 Msamples/sec ADC. A 22 channel with 20 programmable triggers based Direct Memory Access (DMA) controller is used to access all memories and streaming peripheral.

The ARM Cortex-M4 is a 32-bit core having a 3-stage pipeline Harvard architecture and includes an internal prefetch unit that supports speculative branching. The baseline processing system supports single-cycle digital signal processing and SIMD instructions. The ARM Cortex-M0+ coprocessor is an energy-efficient and easy-to-use 32-bit core which is code- and tool compatible with the Cortex-M4 master core. The coprocessor supports the maximum of 100 MHz performance with a single instruction set and reduced code size.

### 5.2. Zynq based ISSA

The *Zynq based ISSA* uses architecture already described in Section 4 and shown in Fig. 9. The *Processing System* 4.5 of ISSA is interfaced with a Zynq-7000 System on Chip multi-core system. The ISSA *Memory System* allocates 200 KB *Scratchpad* and 20 KB *Program* memories, 512MB of SDRAM-DDR3 *Main Memory* with 1 Gbps bandwidth. The ISSA *Scheduler* performs real-time hardware scheduling of multiple processing cores and sensors. The ISSA is subdivided into *I-MCS* and *I-SASHA* based on modes of processing.

#### 5.2.1. I-MCS

The *I-MCS* uses a dual-core ARM Cortex-A9 processor operating at maximum of 650 MHz clock. Each ARM processor has its own Single Instruction Multiple Data (SIMD) media processing engine, memory management unit (MMU), and separate 32 KB level-one (L1) instruction and data caches. Each Cortex-A9 processor provides two 64-bit AXI master interfaces for independent instruction and data transactions to the Snoop Control Unit (SCU).

#### 5.2.2. I-SASHA

The *I-SASHA* uses FGPA based SASHA cores as processing system. The *I-SASHA* uses Xilinx 7-series FPGA having 28K logic cells and 80 DSP blocks. The abstract level design of ISSA Programming Toolkit allows to instantiate multiple copies of the *Test Algorithms* (Discussed in Section 3) on the *I-SASHA* without addressing clocks and technology constraints.

## 6. Results and discussion

The results of different experiments conducted on Interferometry Sensor Systems by executing the *Test Algorithms* (see Section 3.3) are presented below. The section is further subdivided into four subsections; the *Experimental Setup*, the *Single Application Performance*, the *System Throughput* and the *Dynamic Power and Energy*.

### 6.1. Experimental setup

The Interferometry Systems are experimentally tested for the *Test Algorithms* by using the experimental laboratory setup shown in Fig. 7. The target is mounted on a mechanical wave driver PASCO SF-9324 having a frequency range of 0.1 Hz to 5 kHz. The function generator (GWINSTEK AFG-2225) is used to excite the target at a frequency of 80 Hz with a peak to peak amplitude of 5 µm. To obtain the SM signal, a DL7140 laser diode is used (shown in Fig. 7 (c)) having an emission wavelength of 758 µm, the threshold current of 50 mA and with the optical output power of 80 mW. The oscilloscope (GWINSTEK GDS-2204E) is attached with the sensor electronic circuitry to observe the generated SM signal.

The displacement retrieval results are compared from a reference commercial PZT sensor from Physik Instrument (P753.2CD) having 2nm accuracy. The retrieved displacements for a target motion of 80 Hz and its comparison with reference PZT sensor is shown in Fig. 10. Figs. 10 (a) and (b) show SM signal and retrieved target displacement using PZT sensor respectively. Figs. 10 (c) and (d) show the displacement retrieval using the CSU test algorithm and its error comparison with the PZT sensor. The CSU retrieves the target displacement with RMS error of 50 nm which is equivalent to an accuracy of $\lambda/15$. The error is calculated by taking the difference of retrieved displacement using CSU (shown in Fig. 10 (c)) and reference PZT sensor displacement (Fig. 10 (b)). Similarly, Figs. 10 (e) and (f) show the displacement retrieval using the TFSP algorithm and its error comparison with the PZT sensor. The TFSP gets 15.5 nm of RMS error while retrieving the target displacement with an accuracy of $\lambda/50$. Fig. 10 (g) and (h) show the displacement retrieval using IPUM test algorithm and its error comparison with the PZT sensor. The IPUM retrieves the target displacement with RMS error of 12 nm which is equivalent of an accuracy of $\lambda/65$.

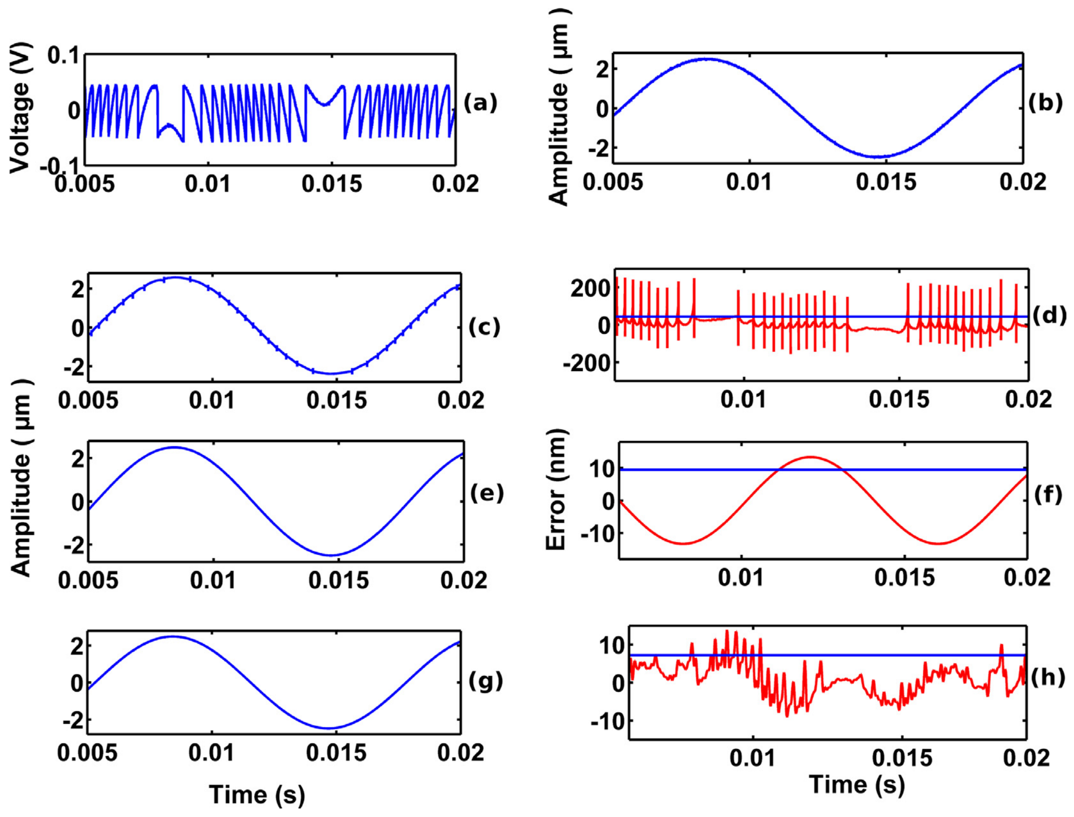


**Fig. 10.** Experimentalresults: (a) Experimentallyacquired SM signal (b) Displacement measured by the reference PZT sensor (c) Displacementretrieval from the CSU algorithm (d) CSU error with respect to the reference PZT sensor (e) Displacementretrieval from the TFSP algorithm (f) TFSPerror with respect to the reference PZT sensor (g) Displacement retrieval from the IPUM algorithm (h) IPUM error with respect to the reference PZT sensor .

**Table 5**
*I-SASHA*, *I-MCS* and *B-MCS*:test applications execution time in milliseconds.

| Algorithms | I-SASHA | I-MCS | B-MCS |
|---|---|---|---|
| CSU | 19.2 ms | 98.2 ms | 145 ms |
| TFSP | 188.3 ms | 556 ms | 854 ms |
| IPUM | 252.8 ms | 1025.2 ms | 1644 ms |

**Table 6**
ISSAspeedups.

| | CSU | TFSP | IPUM |
|---|---|---|---|
| I-SASHA against I-MCS | 5.11 | 2.94 | 4.05 |
| I-SASHA against B-MCS | 7.55 | 4.53 | 6.50 |
| I-MCS against B-MCS | 1.47 | 1.53 | 1.60 |

### *6.2. Single application performance*

In this section, we implement and evaluate the described *Test Algorithms* on *I-SASHA*, *I-MCS* and *B-MCS* and gather results. As the maximum operating frequency of Baseline System is 100 MHz, therefore, we execute the *I-MCS* and *I-SASHA* at the same frequency. In order to test the application algorithms we used 32K words of input SM dataset. The results are further subdivided into *System Performance* and *Performance Comparison*.

#### *6.2.1. System performance*

In this section we measure the time in milliseconds taken by *I-SASHA*, *I-MCS* and *B-MCS* while executing a single *Test Algorithm* (see Table 5) at a time. The algorithms for *I-MCS*, *B-MCS* and *I-SASHA* are designed in software and hardware using C++ and HDL languages, respectively.

While executing CSU algorithm on the *I-SASHA*, it takes 19.2 ms to complete the application. The TFSP on the *I-SASHA* takes 188.3 ms. The IPUM on *I-SASHA* takes 252.2 ms. While executing the *Test Algorithms* on *I-MCS*, the CSU, TFSP, and IPUM algorithms take 98.2, 555.3 and 1025.2 ms of time, respectively. While executing the CSU, TFSP, and IPUM applications on the *B-MCS*, the results show that the *B-MCS* takes 145, 854, and 1644 ms, respectively.

The reason of fast execution of CSU algorithm is that it has regular instructions and data access patterns which can be pipelined easily. The TFSP uses FFT and IFFT applications which are compute intensive but their access patterns are predictable. The IPUM consumes maximum time because of its complex data dependencies and irregular access patterns. The algorithm involves many branch instructions with data access patterns dependencies which generate processing and task communication delay.

The ISSA improves the performance of the algorithms by managing the data access using the data access descriptors. The ISSA descriptor accesses data from the *Front-end Interface* and automatically stores it to local memory from where processing core perform execution. Whereas, the baseline system access data in generic formate using multiple load/store or direct memory access instructions which generate delays and put processing system in idle state.

#### *6.2.2. Performance comparison*

In this section we compare the performance of *I-SASHA I-MCS* and *B-MCS* (shown in Table 6) against one another. While performing real-time signal processing the result shows that the ISSA front-end interface accesses 1 million samples per seconds from a single sensor source. Therefore, the ISSA measures maximum in-

**Table 7**
System performance.

| | I-MCS | I-SASHA | B-MCS |
|---|---|---|---|
| Processing (Opr/s) | $628.22 \times 10^6$ | $1.4 \times 10^9$ | $103 \times 10^6$ |
| Bus (bits/s) | $3.2 \times 10^9$ | $4.2 \times 10^9$ | $512 \times 10^6$ |
| Memory (bits/s) | $1.8 \times 10^9$ | $2.4 \times 10^9$ | $256 \times 10^6$ |
| Front-end Interface (bits/s) | $256 \times 10^6$ | $256 \times 10^6$ | $60 \times 10^6$ |
| Parallel Sensor | 16 | 16 | 5 |

**Table 8**
Dynamicpower and energy consumption for CSU algorithm.

| | Dynamic power (Watt) | Energy (Joules) |
|---|---|---|
| BIS-MCS | 1.47 | 6.82 |
| ISSA-MCS | 1.65 | 5.18 |
| ISSA-SASHA | 1.24 | 0.76 |

stantaneous velocity of 13.08 mm/s, for a laser diode with 785 nm emission wavelength.

*I-SASHA against I-MCS* The results indicate that the *I-SASHA* executes CSU, TFSP, and IPUM 5.11, 2.94 and 4.05 times faster than the *I-MCS*. The reason of having less speedups of *I-MCS* against *I-SASHA* is that it employs the RISC architecture which executes custom general purpose instructions providing flexibility and task level parallelism. Each *I-MCS* RISC core takes two data inputs and generates one data output. Whereas, *I-SASHA* boosts the performance of applications by utilizing data level parallelism while lowering the hardware resources and power requirements. The *I-SASHA* executes the application in single instruction multiple data architecture. It combines several custom instructions into one instruction and performs computation on a chunk of data; this reduces the instruction decoding and data management time.

*I-SASHA against B-MCS* The results show that the *I-SASHA* executes CSU, TFSP, and IPUM algorithms 7.5, 4.5, and 6.5 times faster, respectively. The reasons of speedups are: the *I-SASHA* efficiently shares the system resource that boosts the *Test Algorithm* execution. The system pipelines the algorithms and buffers the intermediate results in *Scratchpad Memory* using the *Memory System*.

*I-MCS against B-MCS* We compare the *Test Algorithms* results of *I-MCS* against *B-MCS*. The results show that the *I-MCS* improves CSU, TFSP, and IPUM algorithms performance up to 1.47, 1.53 and 1.6 times respectively, against the *B-MCS*. The reason of *I-MCS* speedups are: the ISSA maps data transfers in patterns using the *Program Memory* which reduces the *Front-end Interface* input output throughput requirement and *Main Memory* read latency. The ISSA *Scheduler* performs automated run-time resource aware scheduling which handles the application tasks and avoids task scheduling delays and provides fairness.

### 6.3. System throughput

In this section, we measure the I-MCS, I-SASHA and B-MCS real-time throughput while executing the *Test Algorithms* concurrently. The *System Throughput* includes processing system operations per second (Opr/s), bus system, memory system and sensors data rate in bits per second (bits/s) shown in the Table 7. While running all algorithms together the results show that, the I-MCS and I-SASHA *Front-end Interface* can read data from 16 parallel sensors with total data rate 256 M bits per second, whereas the B-MCS can support upto 5 parallel sensors. The I-MCS and I-SASHA *Memory and Bus* systems handle data between processing system and multi sensors with high throughput. The results show that I-MCS and I-SASHA process the algorithms 6.09 × and 13.59 × times faster than the B-MCS, respectively.

The reasons of high speedup are: the ISSA provides function calls which manages data transfer information in descriptors. At run-time the ISSA *Front-end Interface* accesses the data from multiple sensors independently and places it to *Scratchpad Memory*. The ISSA *Scratchpad Memory* manages the complex data patterns of multiple sensors and provide it to the processing cores. Whereas the baseline system uses a processing core to access data from the sensors and perform memory management. The ISSA performs data transfer, memory management and processing in parallel with each other.

### 6.4. Dynamic power and energy

The *Dynamic Power* and *Energy* of *I-MCS*, *I-SASHA* and *B-MCS* was measured (shown in Table 8), while executing the *Test Algorithms* with 100 MHz clock frequency and 1M words of SM input data set. For the power measurement, the ISSA and NXP systems use on-board real-time current sensors and Monsoon Power Monitor, respectively. The Static Power of ISSA is 32 mW where as *B-MCS* consumes 5.8 mW respecively. While comparing, the *Dynamic Power* results show that *I-MCS* draws 12.2% more power than *B-MCS*, respectively. Whereas while comparing the *Energy*, *I-MCS* consumes 1.31 times less *Energy* than *B-MCS*, respectively. While comparing the dynamic power and *Energy* of *I-SASHA* against *I-MCS*, the results show that *I-SASHA* draws 15.6% less *Dynamic Power* and 8.9 times less *Energy*.

The reasons of consuming less *Dynamic Power* and improvement in the *Energy* of *I-MCS* and *I-SASHA* are: ISSA allows processing system to execute application efficiently by reordering multiple data transfers in pattern and schedules them efficiently; this reduces the bus switching and arbitration which improves the application performance and reduces the power consumption.

## 7. Conclusion

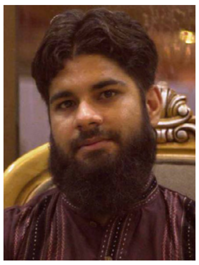
In this work, we have presented the development of Interferometry Sensor System Architecture *ISSA*. The ISSA supports multiple general purpose RISC and application specific hardware accelerator that gives *hardware software co-design* and *parallel processing* approach. The system gives *high-performance* and processes application in *real-time* by using a multiple general purpose RISC processors, *reprogrammable* applications specific hardware. The ISS *Memory System* manages compute intensive and time-critical data and their complex data transfers. The system uses *Front-end Interface* which draws low-power and handles multiple sensor units and their data transfers. The ISSA *Scheduler* gives *energy efficacy* and *high speed* by handling multiple sensors and processing cores without the support of extra hardware or operating system. To program the sensor algorithms ISSA uses a *Programming Toolkit* which is *easy to program* and supports *parallel programming*. In order to prove that our system is suited for sensing applications, we used three different algorithms of varying complexity and accuracy in order to retrieve displacement and vibrational information from a SM interferometric sensor. When compared to a NXP based Baseline Interferometry Sensor Multi-Core System, the ISSA improves the application performance upto 7.5 times, draws 15.6% less dynamic power and consumes 8.9 times less energy.

It may be appreciated that generic, easy-to-program, scalable, and high performance architecture, capable of being easily interfaced with multiple sensors, can not only allow real-time sensing but also enables the deployment of massively parallel sensors performing 1D or 2D sensing such as in imaging of fluid flow rate or modal analysis of mechanical structures for their quality and safety testing.

## References

[1] R. Gensh, A. Aalsaud, A. Rafiev, F. Xia, A. Iliasov, A. Romanovsky, A. Yakovlev, Experiments with Odroid-XU3 Board, Newcastle University, Computing Science, Claremont Tower, Claremont Road, Newcastle England, 2015.
[2] S.N. Agathos, A. Papadogiannakis, V.V. Dimakopoulos, Targeting the parallella, in: Proceedings of the European Conference on Parallel Processing, Springer, 2015, pp. 662–674.
[3] LPC5410x 32-bit arm Cortex-M4/M0+ microcontroller, NXP Semiconductors, 2 July 2015.
[4] MSP432P401x mixed-signal microcontrollers, Texas Instruments Incorporated, 2015.
[5] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, J. Van Dyken, The stratix 10 highly pipelined FPGA architecture, in: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, 2016, pp. 159–168.
[6] L.H. Crockett, R.A. Elliot, M.A. Enderwitz, R.W. Stewart, The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc, Strathclyde Academic Media, 2014.
[7] Y.L. Lim, R. Kliese, K. Bertling, K. Tanimizu, P. Jacobs, A.D. Rakić, Self-mixing flow sensor using a monolithic VCSEL array with parallel readout, Opt. Express 18 (11) (2010) 11720–11727.
[8] L. Esteban, M. Sánchez, J.A. López, P. Kornejew, M. Hirsch, O. Nieto-Taladriz, Development of efficient FPGA-based multi-channel phase meters for ir-interferometers, IEEE Trans. Nuclear Sci. 58 (4) (2011) 1562–1569.
[9] C. Wang, FPGA-based, 4-Channel, High-Speed Phasemeter for Heterodyne Interferometry, University of Rochester. Department of Electrical and Computer Engineering, 2013.
[10] S. Strube, G. Molnar, H.-U. Danzebrink, Compact field programmable gate array (FPGA)-based multi-axial interferometer for simultaneous tilt and distance measurement in the sub-nanometre range, Meas. Sci. Technol. 22 (9) (2011) 094026.
[11] L.A. Vera-Salas, S.V. Moreno-Tapia, A. Garcia-Perez, R.d.J. Romero-Troncoso, R.A. Osornio-Rios, I. Serroukh, E. Cabal-Yepez, FPGA-based smart sensor for online displacement measurements using a heterodyne interferometer, Sensors 11 (8) (2011) 7710–7723.
[12] K.-S. Isleif, O. Gerberding, S. Köhlenbeck, A. Sutton, B. Sheard, S. Goßler, D. Shaddock, G. Heinzel, K. Danzmann, Highspeed multiplexed heterodyne interferometry, Opt. Express 22 (20) (2014) 24689–24696.
[13] S. Merlo, S. Donati, Reconstruction of displacement waveforms with a single-channel laser-diode feedback interferometer, IEEE J. Quantum Electron 33 (4) (1997) 527–531.
[14] M. Norgia, D. Melchionni, A. Magnani, A. Pesatori, High-speed self-mixing laser distance sensor, in: Proceedings of the Eleventh International Conference On Vibration Measurements by Laser and Noncontact Techniques-Aivela: Advances And Applications, 1600, AIP Publishing, 2014, pp. 422–425.
[15] M. Norgia, A. Pesatori, S. Donati, Laser diode for flow-measurement, in: Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC), IEEE, 2015, pp. 1391–1394.
[16] F. Cavedo, A. Pesatori, M. Norgia, P. di Milano, G.E. Solari, Laser rangefinder for steel pipes characterization, in: Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC), IEEE, 2015, pp. 1387–1390.
[17] A. Magnani, A. Pesatori, M. Norgia, Real-time self-mixing interferometer for long distances, IEEE Trans. Instrum. Meas. 63 (7) (2014) 1804–1809.
[18] A. Magnani, A. Pesatori, M. Norgia, Self-mixing vibrometer with real-time digital signal elaboration, Appl. Opt. 51 (21) (2012) 5318–5325.
[19] T. Taimre, M. Nikolić, K. Bertling, Y.L. Lim, T. Bosch, A.D. Rakić, Laser feedback interferometry: a tutorial on the self-mixing effect for coherent sensing, Adv. Opt. Photonics 7 (3) (2015) 570–631.
[20] S. Donati, Developing self-mixing interferometry for instrumentation and measurements, Laser Photonics Rev. 6 (3) (2012) 393–417.
[21] O.D. Bernal, U. Zabit, T. Bosch, Classification of laser self-mixing interferometric signal under moderate feedback, Appl. Opt. 53 (4) (2014) 702–708.
[22] O.D. Bernal, U. Zabit, T. Bosch, Study of laser feedback phase under self-mixing leading to improved phase unwrapping for vibration sensing, Sens. J. IEEE 13 (12) (2013) 4962–4971.
[23] S. Ottonelli, F.D. Lucia, M.D. Vietro, M. Dabbicco, G. Scamarcio, F.P. Mezzapesa, A compact three degrees-of-freedom motion sensor based on the laser–self-mixing effect, Photonics Technol. Lett. IEEE 20 (16) (2008) 1360–1362.
[24] O.D. Bernal, U. Zabit, T.M. Bosch, Robust method of stabilization of optical feedback regime by using adaptive optics for a self-mixing micro-interferometer laser displacement sensor, Sel. Top. Quantum Electron. IEEE J. 21 (4) (2015) 336–343.
[25] C. Bes, G. Plantier, T. Bosch, Displacement measurements using a self-mixing laser diode under moderate feedback, IEEE Trans. Instrum. Meas. 55 (4) (2006) 1101–1105.
[26] U. Zabit, O.D. Bernal, T. Bosch, Time-frequency signal processing for a self-mixing laser sensor for vibration measurement, in: Proceedings of the IEEE Sensors, IEEE, 2012, pp. 1–4.
[27] A.L. Arriaga, F. Bony, T. Bosch, Real-time algorithm for versatile displacement sensors based on self-mixing interferometry, Sens. J. IEEE 16 (1) (2016) 195–202.
[28] Y. Fan, Y. Yu, J. Xi, J.F. Chicharo, Improving the measurement performance for a self-mixing interferometry-based displacement sensing system, Appl. Opt. 50 (26) (2011) 5064–5072.
[29] T. Hussain, O. Palomar, A. Cristal, O. Unsal, E. Ayguady, M. Valero, Advanced pattern based memory controller for FPGA based applications, in: Proceedings of the International Conference on High Performance Computing & Simulation, ACM, IEEE, 2014, p. 8.
[30] T. Hussain, et al., PPMC: A Programmable Pattern based Memory Controller, ARC, 2012.
[31] T. Hussain, et al., Recongurable Memory Controller with Programmable Pattern Support, HiPEAC WRC, 2011.
[32] T. Hussain, O. Palomar, A. Cristal, O. Unsal, E. Ayguady, M. Valero, MAPC: memory access pattern based controller, in: Proceedings of the Twenty Forth International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2014.
[33] NXP Semiconductors, Cortex-M4 MCUs with cortex-M0 co-processors.

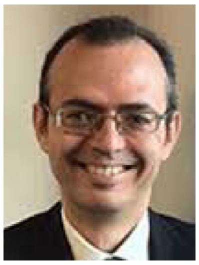
**Tassadaq Hussain** received Ph.D. degree in computer architectures at the Universitat Politécnica de Catalunya (UPC) in collaboration with Barcelona Supercomputing Center and Microsoft Research Center (BSCMSRC). He obtained MSc (Electronics) degree in 2009 from the Institut Supérieur d'Electronique de Paris France. He worked for Infineon Technology digital design department south France. During the stay in Infineon, he worked over Ultra-low Cost Mobile Base Band Chips. From September-2009 to December-2014. He is working as Assistant Professor at Riphah International University Islamabad and serving Unal Color of Education Research and Development as Research Director. His main research interests include Supercomputing for Artificial Intelligence and design of heterogeneous multi-core architectures with the focus on efficient hardware resource scheduling, data and access patterns management strategies for HPC applications.



**Saqib** received his Master's degree in Electrical Engineering with majors in Signal Processing from the Riphah International University (RIU), Islamabad, Pakistan in 2016. He is currently a faculty member with RIU, and also pursuing his Ph.D. degree in electrical engineering with majors in Signal Processing. His area of interests includes FPGA implementation and signal processing for self-mixing interferometry System.

**Usman Zabit** received the Ph.D. degree from Institut National Polytechnique Toulouse (INPT), France, in 2010. He held a post-doctoral position with LAAS-CNRS, Toulouse, working on real-time embedded sensor design. He is a recipient of the Prix Leopold Escande and the European Mechatronics Award 2010 from INPT and European Mechatronics Meeting 2010, respectively.

**Olivier D. Bernal** received the M.Sc. degree in electrical engineering and the Ph.D. degree from Institut National Polytechnique Toulouse (INPT), Toulouse, France, in 2003 and 2006, respectively. He joined the Laboratory of Optoelectonics and Embedded Systems, LAAS-CNRS in 2009 and INPT, where he is currently an Assistant Professor. His main research interests are in the design of analog circuit design for optoelectronics and space applications.



**Thierry Bosch** is a Professor with Institut National Polytechnique Toulouse-ENSEEIHT and Head of the Optoelectronics for Embedded Systems Research Group, LAAS-CNRS. His research interests include laser industrial instrumentation development, including range finding techniques, vibration and velocity measurements. He is Associate Editor of the International Journal of Smart Sensing and Intelligent Systems.