UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria “Leonardo da Vinci”

Corso di Dottorato di Ricerca in
INGEGNERIA DELL'INFORMAZIONE
(SSD: Ing-Inf-01)

Tesi di Dottorato di Ricerca

Design of embedded digital systems for data handling and elaboration in industrial and scientific applications

Autore:

Gianluca Borgese _______________

Relatori:

Prof. Luca Fanucci _______________

Prof. Calogero Pace _______________

Anno 2013

SOMMARIO

Oggigiorno in molti settori di ricerca sia di tipo industriale che scientifico, l'elaborazione e il trattamento veloce ed efficiente dei dati risulta essere sempre più importante, soprattutto vista la sempre più crescente espansione dei sistemi che gestiscono una grande mole di dati multimediali in real-time. Con elaborazione e trattamento dei dati si può intendere un numero elevato di possibili manipolazioni di dati (calcolo logico-aritmetico, compressione, filtraggio numerico, memorizzazione, trasmissione, etc) svolte da uno o più sistemi hardware/software nell'ambito di differenti applicazioni. Nella prima parte di questo lavoro di tesi si introdurranno i concetti alla base dell'elaborazione e del trattamento dei dati e si presenteranno le varie tipologie di sistemi "Embedded", soffermandoci principalmente sulle tipologie trattate in questo lavoro di tesi. Successivamente si tratteranno nello specifico le tre piattaforme embedded progettate: la piattaforma inertial/GPS basata su due microcontrollori a 8-bit per l'acquisizione dei dati provenienti da più sensori (termometri, accelerometri e giroscopi) e la gestione di altri dispositivi (modulo GPS e modulo Zigbee); la piattaforma di emulazione basata su un FPGA per svolgere test funzionali su delle interfacce di comunicazione seriale basate su un nuovo protocollo, denominato FF-LYNX, modellizzate dapprima in System-C e poi implementate su FPGA, da impiegare nell'ambito degli esperimenti di fisica delle alte energie; la piattaforma di calcolo DCMARK basata sempre su FPGA per risolvere velocemente equazioni differenziali non lineari che sono alla base dei sistemi dinamici complessi impiegando un approccio basato sulle CNNs (Cellular Neural Network).Tutti i lavori hanno portato a risultati scientifici interessanti nell'ambito della progettazione di dispositivi embedded. Nella fattispecie, il sistema inertial/GPS ha dimostrato di essere un'efficiente piattaforma impiegabile in differenti ambiti come la body motion recognition, la fall detection, la fotogrammetria aerea, la navigazione inerziale, etc e nuovi spunti potranno nascere in seguito all'integrazione spinta del sistema; Il sistema di emulazione che ha permesso di validare e verificare il corretto funzionamento delle interfacce di comunicazione del protocollo FF-LYNX, risultando essere un mezzo di indagine molto più veloce ed affidabile delle simulazioni ad alto livello; il sistema di calcolo DCMARK che sfruttando un innovativo approccio multi-processore, ha consentito di affrontare la risoluzione di equazioni differenziali non lineari in tempi anche dieci volte più brevi rispetto ai più moderni processori multi-core.

ABSTRACT

Nowadays, in several research fields, both industrial and scientific, fast and efficient data handling and elaboration is more and more important, especially in view of the more and more growing expansion of systems that manage in real-time a large amount of multi-media data. With data handling and elaboration it is possible to define an big number of data manipulations (arithmetic-logic calculation, compression, numerical filtering, storing, transmission, etc) executed by one or more hardware/software systems within different applications. In the first phase of this thesis work the concepts at the bottom of data handling and elaboration will be introduced and the various typologies of "Embedded" systems will be discussed, focusing mainly over the typologies treated in this thesis work. Subsequently, the three embedded platforms designed will be treated in detail: the inertial/GPS platform, based on two 8-bit microcontrollers, for the acquisition of data coming from many sensors (thermometers, accelerometers and gyroscopes) and for the management of other devices (GPS module and ZigBee module); the emulation FPGA-based platform to conduct functional test on serial communication interfaces, based on the new FF-LYNX protocol, modelized firstly in System-C and then implemented on FPGA, to be used within high energy physics experiments; the FPGA-based calculation platform. named DCMARK, that uses a CNN (Cellular Neural Network) approach to rapidly solve non-linear differential equation at the bottom of complex dynamical systems. All these works brought to interesting scientific results within the design of embedded devices. In particular, the inertial/GPS system demonstrated to be an efficient platform usable in different fields such as body motion recognition, fall detection, aerial photogrammetry, inertial navigation, etc and new ideas may be born after the deep integration of system; the emulation system which allowed to validate and verify the proper working of communication interfaces of FF-LYNX protocol, proving to be an investigation instrument faster and more reliable than high level simulation; the DCMARK calculation system which, using an innovative multi-processor approach, allowed to tackle the solving of non-linear differential equation up to ten times more quickly than the more modern multi-core processors.

INDICE

1. INTRODUCTION

1.1. Data handling and elaboration

In the modern era of technology, in which people are surrounded by a large amount of data from TV, radio, internet, etc, there is a more and more growing need of high performance compact systems to acquire, manage and if necessary to transmit data. An example of this kind of system is a Smart phone which has to manage different type of multimedia data (images, videos, sounds, files, etc) and control several electronic modules such as accelerometer and gyroscope sensors, GPS, Bluetooth and Wi-Fi modules, etc. These embedded devices allow to elaborate in real-time thousands of data using many kind of data handling typologies according to the sort of data. There are several kind of data handling typologies: arithmetic-logic elaboration, data compression, numerical filtering, data storing, data transmission, etc. For each typology a dedicated module exists.

1.1.1. Multi-sensor data fusion

The Multi-sensor data fusion (MSDF) [1] is a typical process of handling and integration of multiple data coming from more sensors (temperature, pressure, radiation, etc) into a consistent, complete, accurate, and useful representation. The resulting information is some sense, better than would be possible when these sources were used individually. The main idea is to build a compact data frame ready to be transmitted to other devices for some post-elaboration processes. The expectation is that fused sensor data is more informative and synthetic than the original inputs. Indeed, a creaming off of input data is necessary to hold just significant information. The use of MSDF allows also, in such applications, to harden and improve the information content. For example, in inertial/GPS data fusion, high-frequency acquired inertial sensor data information, integrates the low-frequency acquired GPS data information, during the periods in which GPS data are not present, guaranteeing a nonstop monitoring of body trajectory. The MSDF has also many other application fields such as geospatial information system (GIS), oceanography, wireless sensor networks, cheminformatics, etc. The

1.1.2. Data transmission

In every communication protocol, both wireless and wired, the significant information to transmit has to be encapsulated into a well-defined data packet which contains also several functional and security fields such as preambles, addresses, security codes, type of packet, cyclic redundancy check (CRC), etc. All these fields guarantee both a proper functionality and a good level of reliability to the detriment of data packet size. So when a packet is received, it is necessary to extract data from the packet. Having a large transmission packet is no good for the throughput of the link. It is a challenge to build a no large data packet with an high security and hardness level.

1.1.3. Arithmetic-logic elaboration

There are a lot of typologies of data elaboration to conduct with an embedded system, among those the arithmetic-logic (AL) calculation is the main typology since it is the base for other elaborations. In every industrial and scientific research field there is the necessity to execute AL calculations. In order to do AL operations it is possible to use various approaches using microcontrollers, FPGA, DSP, etc. An example of AL operations are mainly in algorithms of signal elaboration or numerical equation solving in which there are a lot of simple mathematical operations to be performed repeatedly.

1.2. Embedded systems

For a start, it must be explained what is an embedded system. An embedded system [2] is an applied computer system which is designed to control one or more functions, often with real-time computing constraints. It is formed principally by one or more processing unit for executing the system functions (such as microcontrollers, FPGAs, DSPs, etc), a memory unit for the data storing and some peripherals for communicating with the outside world. One of the first modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory, in 1966. It was considered the riskiest item in the Apollo project as it employed the developed monolithic integrated circuits to reduce the size and weight. The embedding of devices into appliances started before the birth of modern PC. Today, embedded systems are deeply ingrained into everyday life. The main idea at the bottom of embedded systems was encapsulating much of system's functionality in the software that runs in the system, therefore it is possible to upgrade the system, acting on software, without modifying the hardware. An embedded system is dedicated to specific tasks, so it is possible optimizing it to reduce the size and the cost. By contrast, a general-purpose computer is designed to do multiple tasks. With the advent of digital age, the dominance of the embedded systems is increased. Each portable devices such as digital watches, MP3 players, cameras, etc are based on an embedded system, they are widespread in consumer, industrial, commercial and military applications. A proper example of embedded system is a Smart phone which represents a complete platform where several different devices (4G, GPS, Bluetooth modules, etc) are integrated on the same chip or board. In the next paragraphs some kind of embedded systems will be shown, such as systems based on microcontroller, FPGA, DSP and SoC.

1.2.1. Microcontroller-based systems

A simple embedded system can have a microcontroller as main managing unit. A microcontroller [3] is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals, in particular, it has commonly the following features:

1. Central processing unit (from 4-bit to 64-bit processors);
2. Volatile memory (RAM) for data storage;
3. ROM, EPROM, EEPROM or Flash memory;
4. Serial input/output such as serial ports UARTs, I²C, USB, SPI;
5. Peripherals: timers, event counters, PWM generators and watchdog;
6. Analog-to-digital converters, digital-to-analog converters.

A microcontroller can be programmed using mainly C and assembly languages but some high performance version based on ARM technology can host also Linux operating system. In an embedded system there can be one or more microcontrollers which control other devices of system. In Fig. 1 there is an

example of microcontroller-based embedded system, where a single microcontroller is
interfaced with several sensors (humidity, temperature, barometric sensors) and with a Zigbee radio transceiver for wireless communication. All these devices can

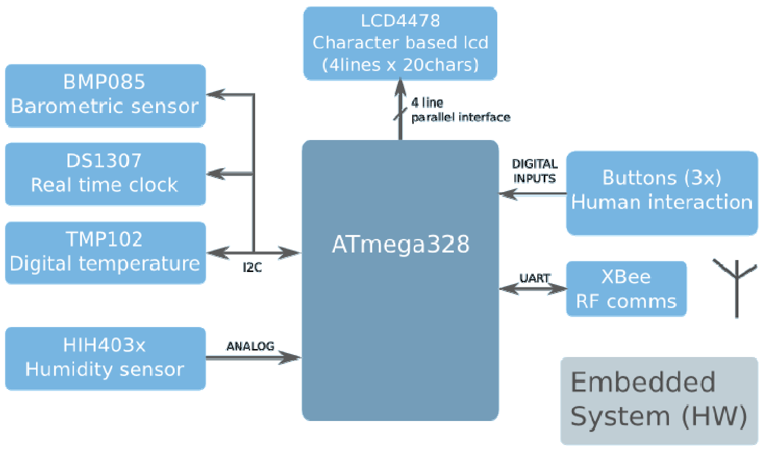


Fig. 1 - Example of microcontroller-based embedded system.

be integrated on a same PCB dual-layer board using SMD components in order to minimize area, volume and weight.

1.2.2. FPGA-based systems

Another kind of embedded system is based on FPGA devices which control whole platform. An FPGA (Field-programmable gate array) [4] is an integrated circuit which can be configured by a designer using a hardware description language (HDL). Today FPGAs have large resources of logic gates and RAM blocks to implement complex digital computations. FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping offer advantages for many applications. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together. Logic blocks can be configured to perform complex combinational functions, or simple logic gates. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. In Fig. 2 it is possible to see block diagram of a FPGA-based embedded system for automotive applications.

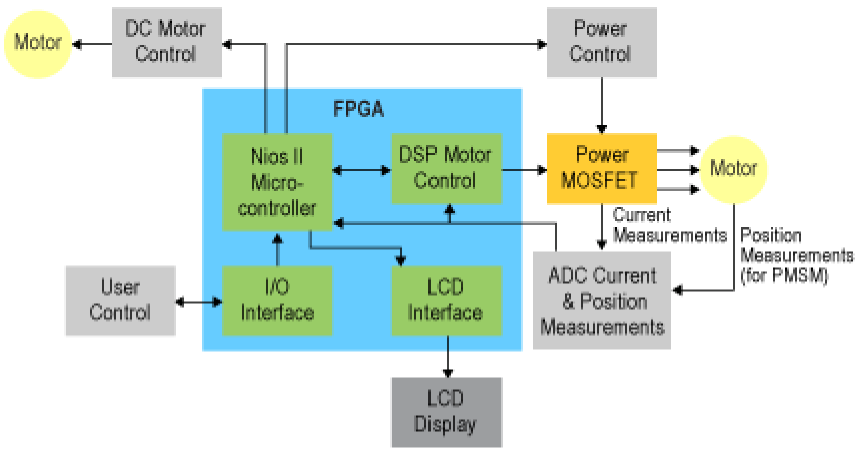


Fig. 2 - Example of FPGA-based embedded system.

1.2.3. DSP-based systems

A DSP (Digital Signal Processor) [5] is special microprocessor which is specialized in digital signal processing. Digital signal processing algorithms typically require a large number of mathematical operations to be performed quickly and repeatedly on a series of data samples. Signals are constantly converted from analog to digital, manipulated digitally, and then converted back to analog form. The architecture of a DSP is optimized specifically for digital signal processing. Fig. 3 shows a typical architecture of an embedded smart sensor based on a DSP.

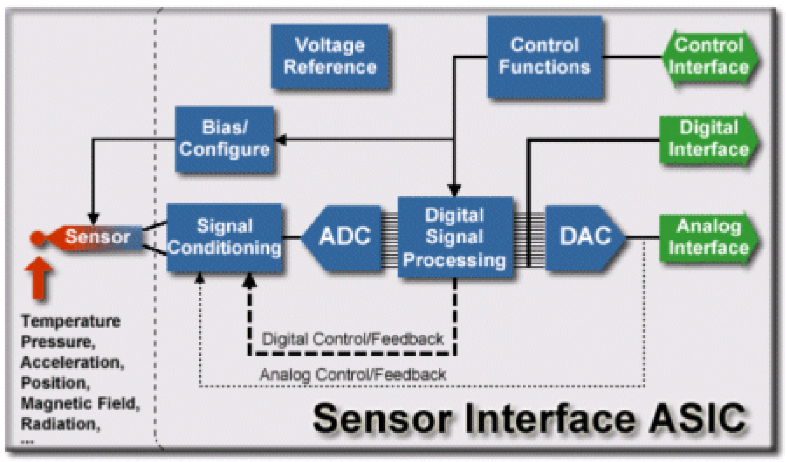


Fig. 3 - Example of DSP-based embedded system (Smart sensor).

1.2.4. System on chip (SoC)

A system on chip (SoC) [6] is an integrated circuit which integrates in a single chip all parts of a computer and often other electronic devices (digital, analog, mixed-signal, radio modules, etc). A system based on SoC technology is a typical embedded system. A microcontroller typically is a single-chip system with no much RAM memory, about hundreds of kB, while a SoC can integrate one or more powerful processors (sometimes multi-core) needing to use large external memory chips (Flash, RAM, EEPROM, etc). This kind of system can run operating system such as Linux or Windows.

A SoC is formed typically by:

1. Microcontroller(s), microprocessor(s) or DSP core(s);
2. Graphics or multi-media processor(s);
3. Memory blocks such as ROM, RAM, EEPROM and Flash memory;
4. Timing sources including oscillators and phase-locked loops;
5. Peripherals including counter-timers, real-time timers and power-on reset generators, radio modules;
6. External interfaces such as USB, FireWire, Ethernet, USART, SPI;
7. Analog interfaces including ADCs and DACs;
8. Voltage regulators and power management circuits.

These blocks are connected by either a proprietary or industry-standard bus. SoCs can be fabricated using several technologies such as full custom, standard cell, FPGA, etc. These systems consume less power and have a lower cost and higher reliability than the multi-chip systems that they replace. In Fig. 4 it is shown a SoC Nvidia Tegra 600-series.

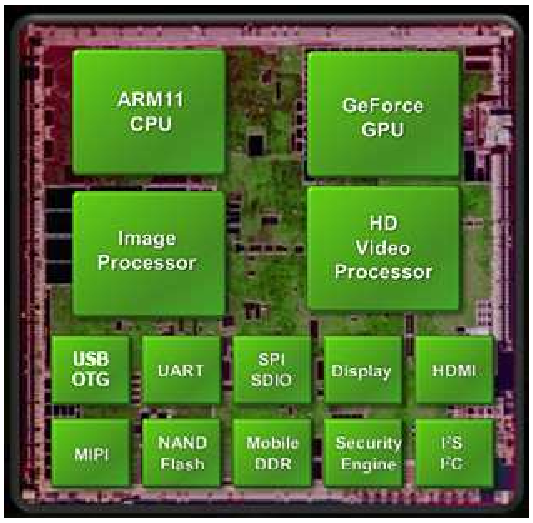


Fig. 4 - Example of SoC Nvidia Tegra 600-series.

2. DEVELOPMENT OF AN INERTIAL/GPS PLATFORM FOR MOTION DATA HANDLING

2.1. *Introduction*

Nowadays, in many research fields such as body motion recognition (BMR), fall detection (FD), aerial photogrammetry (AP), inertial navigation (IN), etc, there is the necessity to acquire and transmit all body motion parameters (axial accelerations, angular rates, global position, speed, etc) by wireless to a remote host system for control and tracking purposes. The possibility to acquire these motion information in a remote real-time way is more and more requested.
With regard to BMR [7], [8] and FD [9], there are several areas of interest (e.g.: 3D virtual reality, biomedical applications, robotics) in which it is extremely important to detect and recognize all or some human body movements and maybe to reproduce those using a robot. AP [10] and IN [11], [12] fields are older than BMR and FD, but we can find a multitude of different and innovative approaches and applications such as pedestrian navigation in harsh environments [13], agriculture automated vehicles [14], or animal motion analysis [15].
In the market there are several kind of systems which are not general purpose but are highly specialized for a particular application. Some systems use high performance and high cost devices, others are not wireless-based or are too heavy. The main idea is to design a low-cost, complete and flexible system with these features and which can be used for several applications. This system should be compact, portable, lightweight and highly integrated.

Table 1 - Comparison with commercial IMUs

IMU Name	HG9900 HoneyWell	MMIMU DraperLab	ADIS16350 AnalogDevices
Typology	Laser	MEMS	**MEMS**
Gyros Bias (1σ) [°/hr]	< 0.003	1	**54**
Gyros Random Walk [° / √hr]	< 0.002	0.030	**4.2**
Accelers Bias (1σ) [mg]	< 0.025	0.1	**0.7**
Accelers Random Walk [m/s/ √hr]	0.0143	0.035	**2.0**
Acc_Bias Pos. Error (1hr) [km]	~ 1.59	~ 6.35	**~ 44.5**
Size [mm]	139.7 x 162.6 x 135.6	68.5 (Ø) x 35.5 (h)	**23.2 x 22.7 x 23.3**
Weight [kg]	< 3	0.260	**0.016**
Power Consumption [W]	< 10	< 3	**< 0.285**
Price [k$]	**~ 100**	**~ 1.3**	**~ 0.6**

2.2. *System design strategy*

To reach these features it is necessary to design the architecture in a smart way and to select single components in order to save space and to decrease system weight as much as possible. In the first prototype, in order to reduce space and weight, we chose, as inertial sensor, a MEMS inertial measurement unit (IMU) (Analog Devices ADIS16350), constituted by a tri-axial accelerometer and a tri-axial gyroscope. This IMU is a strapdown type system which is intrinsically compact, highly integrated and low-cost but it is not very accurate. In the market

there are many kind of very high performance IMU but they don't respect our trade-off requirements of low space, weight and cost (Table 1). We designed the system on two different planar boards (main and power boards) using a PCB dual layer-fashion approach. The supply battery packet was reduced to only two rechargeable NiMh AAA batteries. With a reduced battery packet, the system energy budget is very important to consider. To generate the necessary voltage levels we needed two high-efficiency switching step-up voltage regulators to convert a 2.4V nominal input voltage in two output voltage levels: 5V and 3.3V. In order to handle many motion data, it is important to exploit the available wireless and wired transmission bands organizing data in easily transmissible short packets.
In addition to hardware system side, a remote C-based graphical user interface (GUI) is installed on an Pc-host to control system operations, set inertial sensor parameters (offset, calibration, alignment, etc), display motion variables progress, track trajectories, shoot photo, etc. A trajectory reconstruction algorithm Kalman-based is implemented in the system software for supporting applications such as inertial navigation or motion parameters detection These data elaborations are conducted on software side, instead of hardware side, in order to reduce the computational load of microcontrollers, to speed system operations up and to obtain an easier data handling using the remote GUI.

2.3. System architecture

As explained previously, this system is designed on two separated boards: Main Board and Power board. The first manages all control operations, acquiring data from inertial sensor and GPS module, sending data packets to the host pc and receiving command packets from inertial system by wireless; the latter provides the two supply voltage levels to main board.

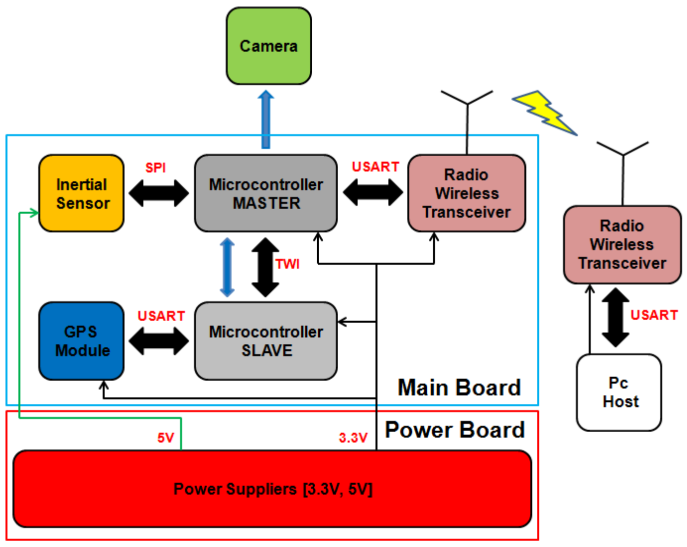


Fig. 5 - Complete inertial/GPS system block diagram.

2.3.1. Power board

The Power board is constituted by two step-up converters (Maxim MAX756) which allow to provide both 5V and 3.3V voltage guaranteeing up to 400mA load current and an efficiency of about 85% (input voltage over 2.2V). As said before, the two rechargeable NiMh battery pack have a nominal voltage of 2.4V and a nominal capacity of about 2650mAh. Considering a nominal input energy of about 6.36Wh against a max required load power of about 625mW (worst case), our system has an autonomy of about 9 hours (experimentally verified).

2.3.2. Main board

The Main board is the control core of whole system. The modules on the board are: two 8 bit microcontrollers (Atmel ATMEGA8) (called Master and Slave), a GPS module (Fastrax UP500), a zigbee transceiver (Maxstream XBEE) and an inertial sensor (AnalogDevice ADIS16350). To support some applications such as aerial or ground photogrammetry, an high resolution camera (Canon SX200IS) was interfaced. Main features of these devices are:

1. *ADIS16350*: is a low-power (165mW @ 5V) complete inertial measurement station. It is constituted by one tri-axial accelerometer, one tri-axial gyroscope and one tri-axial thermometer for thermal compensation. It transfer inertial data with 14 bit resolution, to the output registers, accessible via a 2MHz SPI interface, at a maximum sample rate of 819.2Hz (350Hz bandwidth). The inertial sensors are precision aligned across axes, and are calibrated for offset and sensitivity.
2. *UP500*: is a low-power (90mW @ 3V) GPS receiver module with embedded antenna and fix rate up to 5Hz. Communication is based on NMEA protocols, via RS232 link up to 115.2kbps. It supports WAAS/EGNOS correction to improve position resolution up to about 2m.
3. *XBEE*: is a low-power (165mW @ 3.3V) 2.4GHz transceiver which implements ZigBee^{TM} protocol and has a transmission range of about 80m. Transmission and reception buffers allow efficient data stream packetization, also required to reach the rated communication speed because every data exchange requires the presence of an about 20 bytes long header. It is interfaced through RS232 protocol up to 115.2kbps.

As we can see in Fig.5, Master microcontroller is connected to ADIS16350 through SPI interface, to XBEE through USART interface, to Slave microcontroller through TWI interface and to the high resolution camera by means of one I/O pin.
The Slave microcontroller is connected only to UP500 by means of USART interface and to Master microcontroller as said before. Another Slave I/O pin is used to send an interrupt to Master when a new GPS frame is ready. Master and Slave are clocked with two 14.7654MHz quartz.

2.3.3. High resolution camera

We used a low-cost 12,1Mpixels Canon SX200IS camera (5-60mm lens focus, 4X digital zoom, 12X optical zoom, shutter speed 1s -1/3200s) (Fig. 6a). The firmware was updated with an unofficial version in order to acquire full control of the camera functions. In particular, we exploited the possibility to remotely shoot photos applying a 3V pulse to the USB port, using a BJT, as in Fig. 6b, and to store photos in uncompressed format (RAW), as required for photogrammetry applications [16]. For georeferencing each picture, a progressive number, corresponding to the file number on the memory card, is recorded on the inertial data frame.

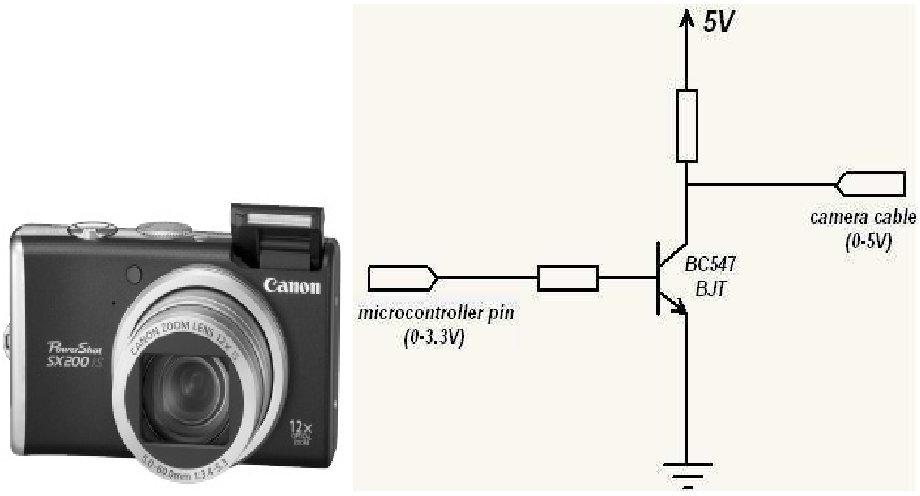


Fig. 6 - Canon SX200i Camera (a), Camera/Microcontroller interfacing (b).

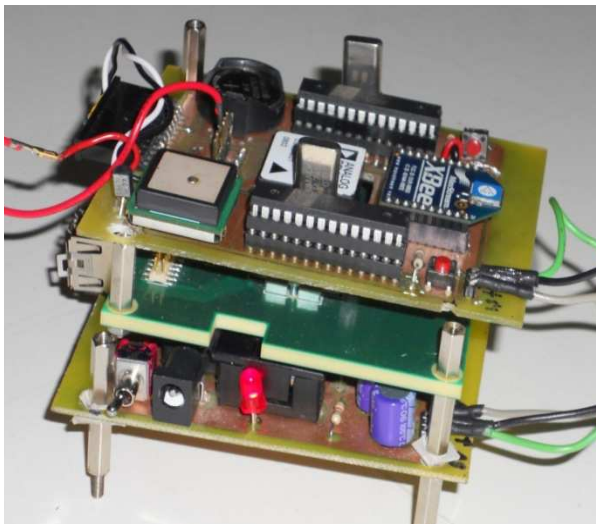


Fig. 7 - First inertial/GPS system prototype v2.2.

2.4. *System working and data protocol*

Thanks to a simple but complete remote GUI, the Pc-host can start every system operation, as will be explained in next sections. There are three kinds of command packets (Fig. 8) that can be sent to the system:

1. operation request (GPS/Inertial data readout, photo shooting, offset readout);
2. configuration setting;
3. configuration readout.

Each command packet is identified by system by means of different opcodes (Fig. 9).

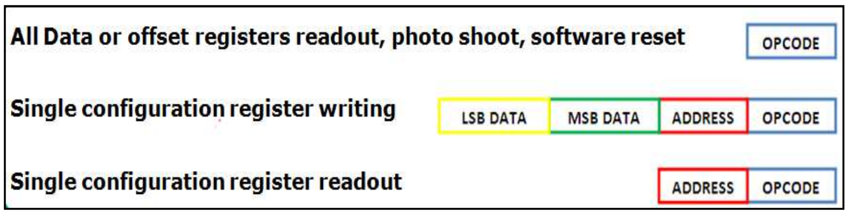
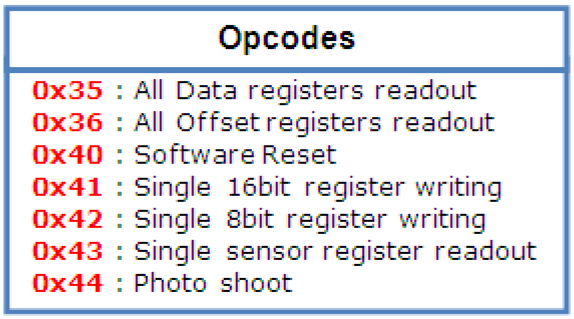

All Data or offset registers readout, photo shoot, software reset
OPCODE
Single configuration register writing
LSB DATA
MSB DATA
ADDRESS
OPCODE
Single configuration register readout
ADDRESS
OPCODE

Fig. 8 - Command structure.



Opcodes

0x35 : All Data registers readout
0x36 : All Offset registers readout
0x40 : Software Reset
0x41 : Single 16bit register writing
0x42 : Single 8bit register writing
0x43 : Single sensor register readout
0x44 : Photo shoot

Fig. 9 - Command opcodes.

The system requirement was to transmit synchronized data from inertial sensor, operating at 100Hz, and GPS module operating at 5Hz (Fig. 10). This inertial sensor sample rate is important to get a good position resolution in case of trajectory tracking calculations. Hence the data stream has to contain 20 inertial frames plus one GPS frame every 200ms.
The inertial data frame is 20 bytes long and contains the following fields: supply voltage, x/y/z temperatures, x/y/z angular rates, x/y/z linear accelerations. The sensor has to be read by the Master every 10ms and this is guaranteed by a dedicated timer.
The most important problem is constituted by the verboseness of GPS data: in fact, NMEA sentences contain hundreds of bytes. So we had to select only the necessary information, otherwise we were able to reach the specified data rate.

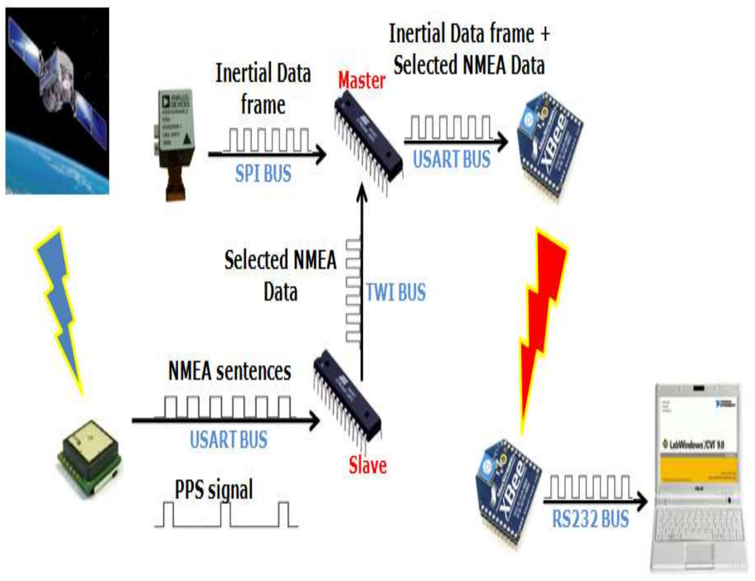


Fig. 10 - Inertial/GPS system operations.

Because we were not interested in all GPS information, at the start up the Slave initializes the GPS module to send only four sentences:

1. GGA: Global Positioning System Fix Data;
2. GSA: GPS DOP and active satellites;
3. VTG: Track Made Good and Ground Speed;
4. RMC: Recommended Mininum Navigation Information.

These NMEA sentences contain main information which can be useful for several different applications.
Moreover, the Slave creams off the received sentences and stores in RAM only the information to display, i.e. a total of 72 bytes.
Even if reduced in this way, the time required to send such information is still too high (about 6.25ms) in order not to compromise the regularity of the inertial sensor reading.
So we decided to divide the GPS answer in 8 packets of 9 bytes and to send, every 20 ms, two inertial frames plus a GPS packet. So, in 200ms, we send 8 frames of 51 bytes (frame number, 2 inertial frames, 1 GPS packet, photo number) and last 2 frames of 42 bytes (frame number, 2 inertial frames, photo number) as shown in Fig.11.
Data acquired from PC are reconstructed, displayed and stored in a text file for further elaboration; GPS data are also processed at run-time to display the trajectory. The frame number is used to identify each frame within a second (50 frames/s) and is used for:

1. reconstruction of GPS information;
2. identification of any frame lost in reception.

Finally, the Photo number allows for the association of picture files in the SD card with time, position and attitude of the camera.
The complete system protocol is better explained in the flow chart in Fig. 12.

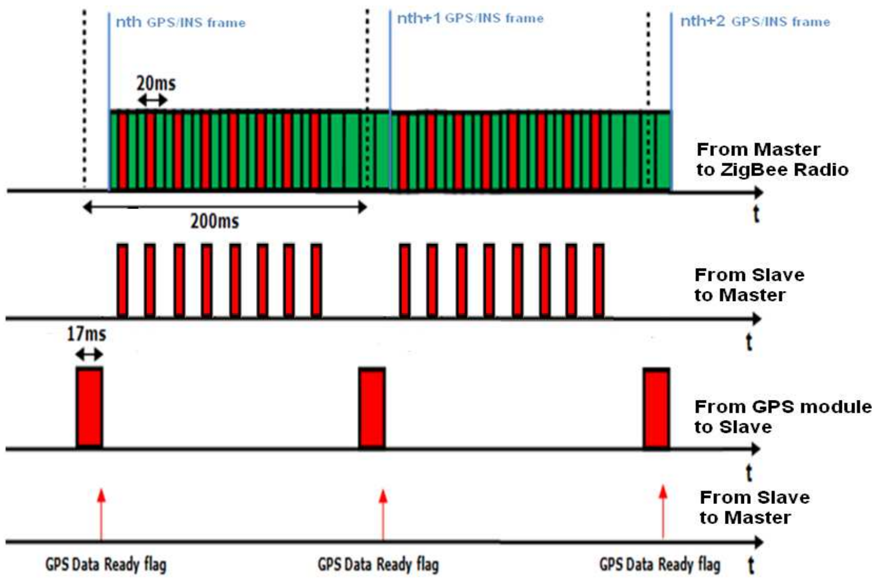


Fig. 11 - GPS/Inertial data timing (in red the GPS data, in green the inertial data).

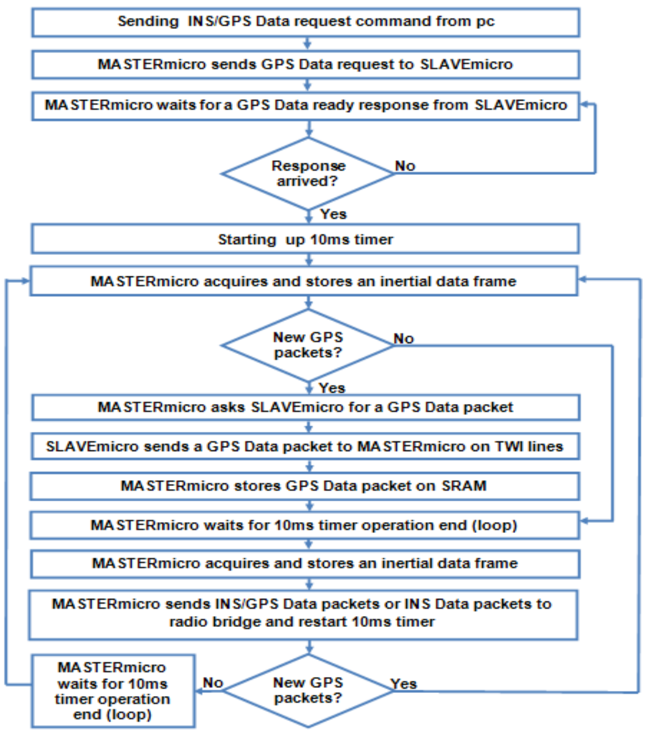

Sending INS/GPS Data request command from pc
MASTERmicro sends GPS Data request to SLAVEmicro
MASTERmicro waits for a GPS Data ready response from SLAVEmicro
Response arrived?
No
Yes
Starting up 10ms timer
MASTERmicro acquires and stores an inertial data frame
New GPS packets?
No
Yes
MASTERmicro asks SLAVEmicro for a GPS Data packet
SLAVEmicro sends a GPS Data packet to MASTERmicro on TWI lines
MASTERmicro stores GPS Data packet on SRAM
MASTERmicro waits for 10ms timer operation end (loop)
MASTERmicro acquires and stores an inertial data frame
MASTERmicro sends INS/GPS Data packets or INS Data packets to radio bridge and restart 10ms timer
MASTERmicro waits for 10ms timer operation end (loop)
No
New GPS packets?
Yes

Fig. 12 - Data protocol flow chart.

After the reception of a data request from host pc, master microcontroller sends a GPS data request to slave microcontroller and waits for response checking the GPS data ready flag. When slave acquires and creams off a GPS frame it sets the GPS data ready flag so that master starts a 10ms timer up and acquires an inertial frame storing it on RAM. Then master asks slave a single GPS packet which is received on TWI line and immediately stored on RAM. When 10ms timer stops, master acquires a second inertial frame storing it on RAM. In the end master sends the two inertial frames and a GPS packet to XBEE module which sent them to host-pc. When these operations are over, master restarts 10ms timer and begin a new operation cycle. If there is an interruption of GPS operations, master continues to send to host-pc only inertial frames respecting the 10ms timing.

2.5. Host data handling

2.5.1. Inertial data elaboration

Data acquired from the inertial sensor can be processed to obtain position and orientation of a body and to track a three dimensional trajectory. This technique is called inertial navigation and it is used in a wide range of applications.
Inertial data are processed following the scheme [17] in Fig.13
where:

1. U_acc: signals from accelerometers;
2. U_omega: signals from gyroscopes;
3. a: linear acceleration;
4. v: linear velocity;
5. ω: angular velocity;
6. C: rotation matrix.

The subscripts *b* denote the body coordinate system (that is the navigation system's reference frame) while the subscripts *n* denote the local coordinate system (in which the body move).
The first step of trajectory reconstruction algorithm is the correction of accelerometers and gyroscopes signals. The correction of errors on signals is the most important step of algorithm, because errors influence overall system performance [18]. In particular, propagation of orientation errors caused by noise, perturbing gyroscope signals, is identified as the critical cause of a body position drift. The main cause of errors are: scale factor, bias, drift, temperature, non-orthogonality. In order to compensate them it is necessary to perform a procedure of calibration. A first coarse calibration was executed using the automatic calibration of ADIS16350 managed from remote GUI software. Then a finer calibration was conducted manually. Among all calibration methods proposed in literature, the most appropriate calibration technique for low-cost sensors is the "*modified multi-position calibration method*" [19], [20]. Its aim is to find all calibration parameters (bias, scale factor, non-orthogonality, etc) of sensors. It consists in laying out sensors in different linearly independent positions in order to define a system of linearly independent equations which outnumbers the set of calibration parameters to find.

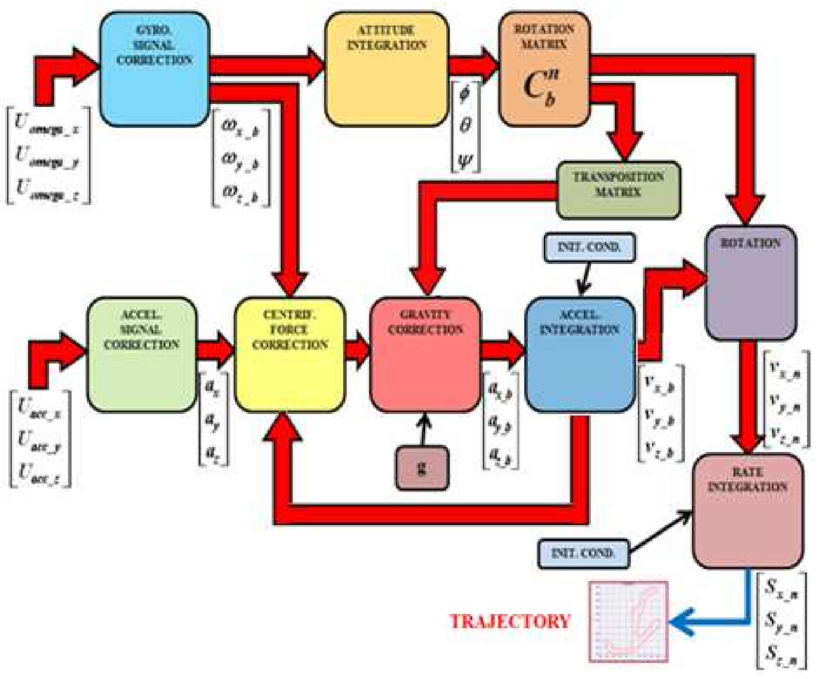


Fig. 13 - Block diagram of the trajectory reconstruction algorithm.

The linear acceleration and angular velocity error can be modeled as:

$$\delta a = b_{acc} + \lambda_{acc} a + c_{acc\_T}(T - T_0) + \nu_{acc} \qquad (1)$$

$$\delta \omega = b_{gyro} + \lambda_{gyro} \omega + c_{gyro\_T}(T - T_0) + \nu_{gyro} \qquad (2)$$

Where:

1. b_{acc} and b_{gyro} are the sensor bias;
2. λ_{acc} and λ_{gyro} are the sensor scale factors;
3. c_{acc_T} and c_{gyro_T} are the sensor thermal constants;
4. ν_{acc} and ν_{gyro} are the sensor measurement noises, $\nu_{acc} = \sigma_{acc} * \sqrt{sample\ rate}$ and $\nu_{gyro} = \sigma_{gyro} * \sqrt{sample\ rate}$, σ_{acc} and $\sigma_{gyro}$are noise densities;
5. T and T_0 are the temperatures during the measurement and at sensor start-up respectively.

In Table 2 there are the calibration parameters obtained according to [19], [22], [23].

Table 2 - Calibration parameters

b_{acc_x}	0.012133g
b_{acc_y}	0.023295g
b_{acc_z}	-0.03593g
b_{gyro_x}	0.3766 °/s
b_{gyro_y}	0.1963°/s
b_{gyro_z}	0.6270°/s
λ_{acc_x}	0.00775
λ_{acc_y}	0.008838
λ_{acc_z}	0.008041
λ_{gyro_x}	0.004818
λ_{gyro_y}	0.004042
λ_{gyro_z}	0.009385
c_{acc_T}	4 mg/°C
c_{gyro_T}	0.1°/s/°C
v_{acc}	1.85mg $\sqrt{Hz}$
v_{gyro}	0.05°/s $\sqrt{Hz}$

After the calibration phase, it is necessary to compensate the centrifugal acceleration and the acceleration of gravity effects obtaining accelerations in body coordinate system . The former is compensated subtracting the vector product between angular velocities (from gyroscopes) and linear velocities (from numerical integration of accelerations), the latter is compensated adding the scalar product between transposed rotation matrix and the gravity acceleration. After a numerical integration velocities in body coordinate system are obtained. In order to pass to local coordinate system the linear velocities are multiplied by the rotation matrix and then are integrated to have body trajectory. The angular velocities are also integrated, obtaining the information about the orientation (Euler angles) and the rotation matrix (for transformation from b-frame to n-frame). The equations to integrate and the rotation matrix are [21]:

$$\dot{\phi} = \left(\omega_{y\_b} \sin\phi + \ \omega_{z\_b} \cos\phi\right) \tan\theta + \ \omega_{x\_b} \qquad (3)$$

$$\dot{\theta} = \left(\omega_{y\_b} \cos\phi - \ \omega_{z\_b} \sin\phi\right) \qquad (4)$$

$$\dot{\psi} = \left(\omega_{y_b} \sin\phi + + \omega_{z_b} \cos\phi\right) \sec\theta \qquad (5)$$

$$C_b^n = \begin{bmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\theta\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} (6)$$

where transformation from reference axes to a new frame is expressed as:

1. rotation through angle ψ about reference z-axis;
2. rotation through angle θ about new y-axis;
3. rotation through angle ϕ about new x-axis.

However, also with a perfect correction of errors, it isn't possible to obtain a great position accuracy for long time using only MEMS IMU but it is necessary to include information from GPS module, integrated in our system. Inertial and GPS modules are complementary: the former is characterized by high measurement frequency but short-term accuracy while the second by long-term accuracy but low measurement frequency. The main idea is to reconstruct trajectory by means of inertial data acquired between two GPS acquisitions and then to correct accumulated errors in inertial data using the stable information from GPS module. The Kalman filter is the most used algorithm for this purpose. In literature there are several implementation of Kalman filter depending on the features of devices [3]. To obtain a correct integration of Inertial and GPS data it is important to have high synchronization between data acquisitions. The implementation of Kalman filtering is included into remote GUI.

2.5.2. GPS data handling

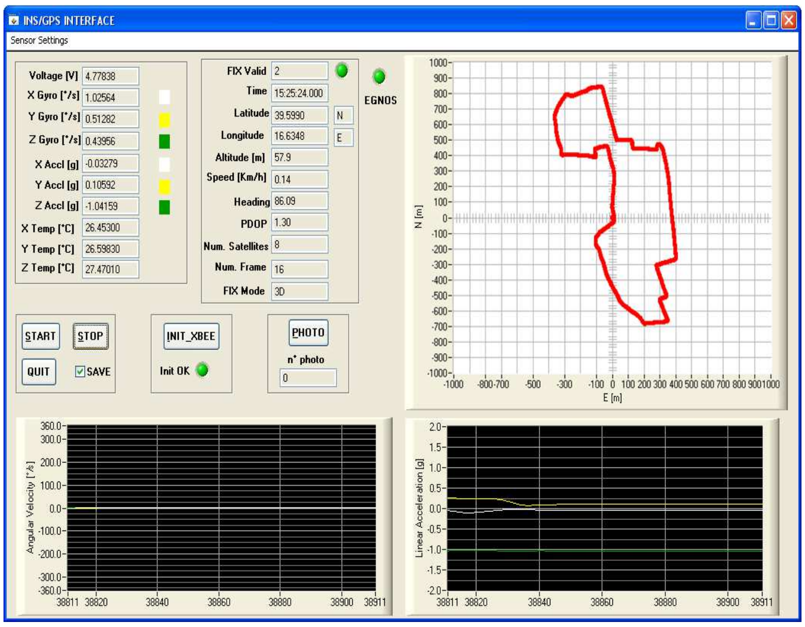
In order to plot a GPS trajectory in a two dimensional graph it is necessary at first to transform GPS geodetic coordinates (*longitude λ, latitude ϕ, height h*) to ECEF (Earth-Centered-Earth-Fixed) coordinates (*Xe, Ye, Ze*) and then to NED (Nord-East-Down) coordinates (*xn, yn, zn*) according to following equations. *N(ϕ)* is the *normal* that is the distance from the surface to the Z-axis along the ellipsoid normal. *a* is the semi-major ellipsoid axis and *e* is the first numerical ellipsoid eccentricity. *Rn/e* is a transformation matrix from ECEF to NED coordinates. *Xer, Yer, Zer* are ECEF reference coordinates.

$$\begin{aligned} X_e &= (N(\phi) + h)\cos\phi\cos\lambda \\ Y_e &= (N(\phi) + h)\cos\phi\sin\lambda \\ Z_e &= (N(\phi)(1 - e^2) + h)\cos\phi\cos\lambda \end{aligned} \qquad N(\phi) = \frac{a}{\sqrt{(1 - e^2\sin^2\phi)}}$$

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = R_{n/e} \times \begin{bmatrix} X_e - X_{er} \\ Y_e - Y_{er} \\ Z_e - Z_{er} \end{bmatrix} \quad R_{n/e} = \begin{bmatrix} -\sin\phi\cos\lambda & -\sin\phi\sin\lambda & \cos\phi \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\phi\cos\lambda & -\cos\phi\sin\lambda & -\sin\phi \end{bmatrix}$$

2.6. *Remote GUI*

The Remote GUI is developed using LabWindows© development environment based on C language. The GUI allows to manage every system operation. As seen in Fig.14 in the window there are three main sections: a graph section to display GPS trajectory, angular velocity and linear acceleration; a boxes section to show inertial sensor parameters (supply voltage, x-y-z linear accelerations, angular velocities and temperatures) and GPS parameters (time, latitude, longitude, altitude above mean sea level, height of geoid above WGS84 ellipsoid, speed, heading and PDOP; a command section to initializing XBEE radio-bridge, to start/stop system operations and to shoot photos. It is also possible to save data into a text file for offline analysis. In the top of window there is a menu in which user can access inertial sensor setting mode and manually change gyroscope dynamics, number of tapes of Bartlett FIR digital filter, sample rate, accelerometer ad gyroscope offset or use automatic procedures of axial alignment, offset compensation, calibration (Fig. 15). The numerical integration algorithm, the Kalman filter and the coordinates transformation are integrated into the GUI.



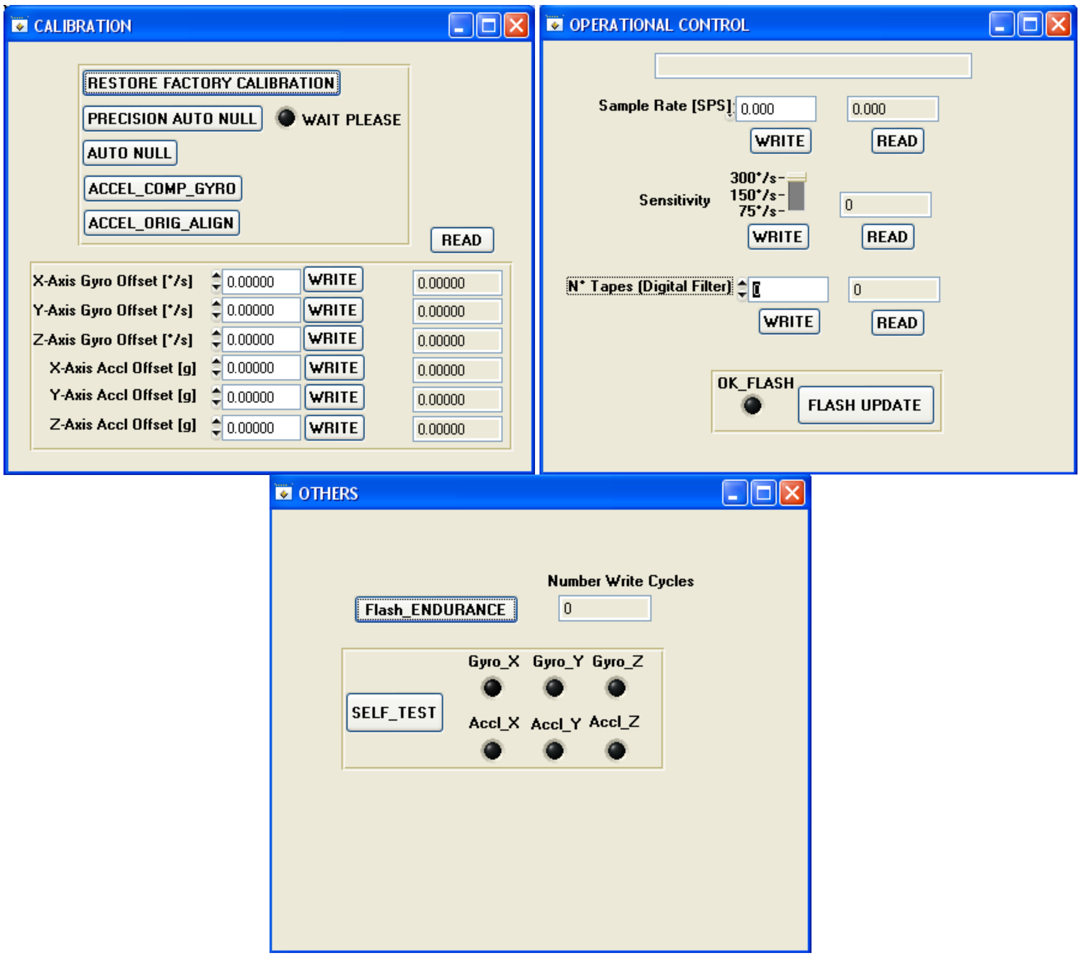
Voltage [V]	4.77838
X Gyro [°/s]	1.02564
Y Gyro [°/s]	0.51282
Z Gyro [°/s]	0.43956
X Accl [g]	-0.03279
Y Accl [g]	0.10592
Z Accl [g]	-1.04159
X Temp [°C]	26.45300
Y Temp [°C]	26.59830
Z Temp [°C]	27.47010

FIX Valid	2	
Time	15:25:24.000	
Latitude	39.5990	N
Longitude	16.6348	E
Altitude [m]	57.9	
Speed [Km/h]	0.14	
Heading	86.09	
PDOP	1.30	
Num. Satellites	8	
Num. Frame	16	
FIX Mode	3D	

☑ SAVE

n° photo: 0

Fig. 14 - System GUI with an example of GPS trajectory.



X-Axis Gyro Offset [°/s] 0.00000 0.00000
Y-Axis Gyro Offset [°/s] 0.00000 0.00000
Z-Axis Gyro Offset [°/s] 0.00000 0.00000
X-Axis Accl Offset [g] 0.00000 0.00000
Y-Axis Accl Offset [g] 0.00000 0.00000
Z-Axis Accl Offset [g] 0.00000 0.00000

Sample Rate [SPS] 0.000 0.000
Sensitivity 0
N° Tapes (Digital Filter) 0 0

Number Write Cycles 0

Fig. 15 - Calibration, operation control and other sub-windows.

2.7. System testing

In order to verify proper working of system many kind of tests are conducted on system modules.

2.7.1. Accelerometers/Gyroscopes test

To test accelerometers and gyroscopes, two type of tests were conducted. In the first test, the system was placed on a strobe speed-controlled turntable with velocity of 33 rpm and 45 rpm, to evaluate biases and the correct angular velocity measured by gyroscopes; in the second test, system was placed on a radio-controlled toy car (Fig. 17) and various movements were performed to test the performance of the whole inertial system (Fig.16).

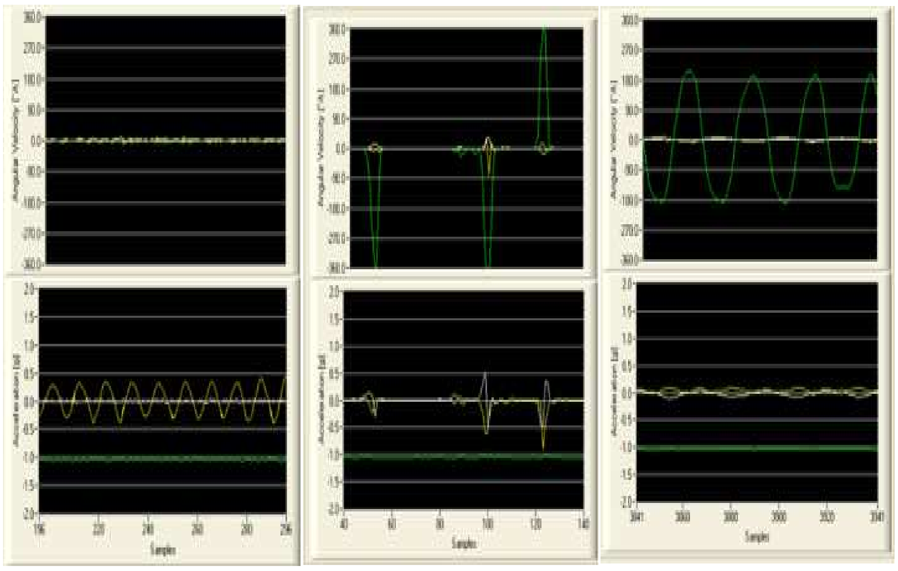


Fig. 16 - Sensor responses for various movements performed (slewing rounds, spins, back/forth).

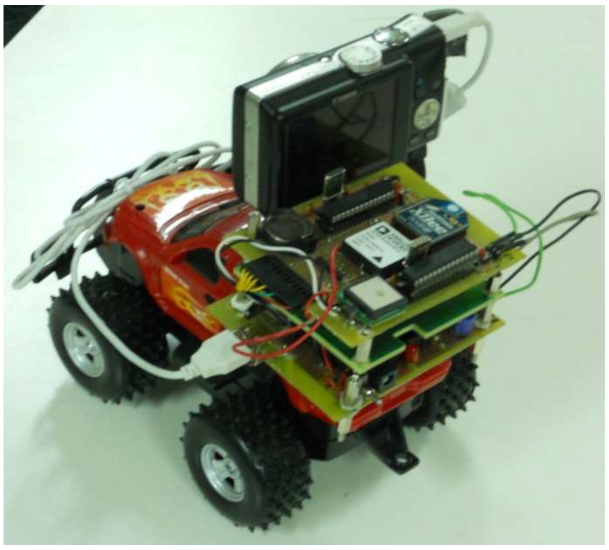


Fig. 17 - System mounted on a toy car for a test.

2.7.2. GPS module test

Moreover, the system was mounted on a car in order to verify GPS module operations and the coordinates transformation algorithm using GPS data (Fig.14). Another kind of test allows to analyze the proper working of GPS module along a closed path and comparing results with a high accuracy differential GPS module. From this test we valued position errors along x, y and z axis using a statistical

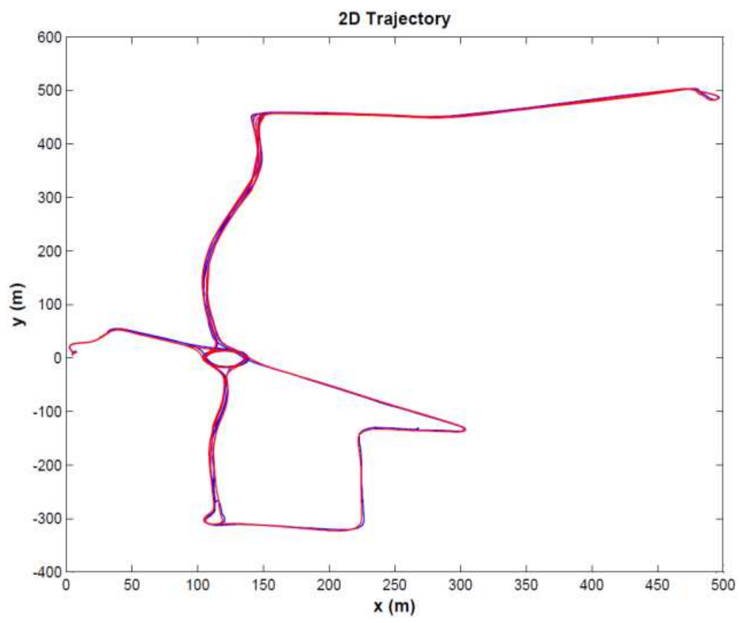


Fig. 18 - Results comparison between our GPS (red) and differential GPS (blue).

analysis. In Fig.18 the trajectory comparison between our GPS module and differential GPS module is shown. In Table 3 and Fig. 19 there are the error distribution parameters. The mean position error is lower than 1m for x and y axis with a standard deviation lower than 2m, only for the z axis the mean position error is of about 5m.

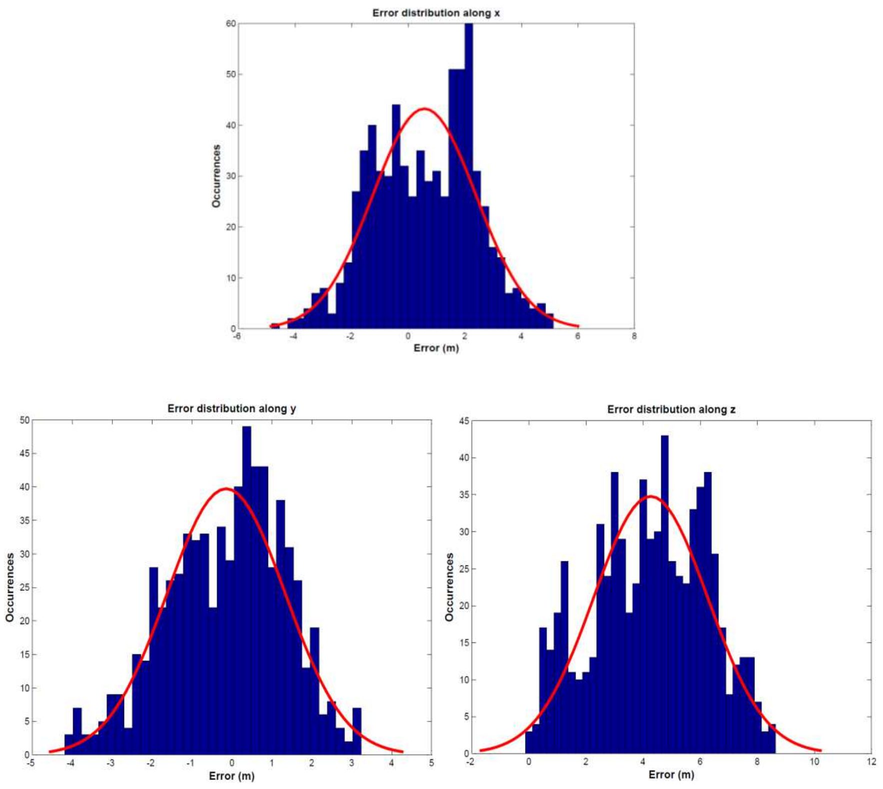


Fig. 19 - X/Y/Z error distributions.

Table 3 - Error distribution parameters

X-error Mean	**0.573m**
Y-error Mean	**-0.143m**
Z-error Mean	**4.267m**
X-error σ	**1.825m**
Y-error σ	**1.480m**
Z-error σ	**1.997m**

2.7.3. Inertial-based trajectory reconstruction test

After the GPS trajectory reconstruction test, we conducted an Inertial-based trajectory reconstruction test to verify the quality of trajectory reconstruction algorithm and of Kalman filtering. For this test the strobe speed-controlled turntable was used. As it can be seen in Fig. 20 using just the reconstruction algorithm, after about 25 loop at 33rpm, there is an increasing of offset and bias which deform the circular trajectory with a spiral divergence; with Kalman filtering the trajectory is very stable and it is evident the decreasing of x/y error as shown in Table 4 with respect to Table 5 without Kalman filter. In Fig. 21 the x/y position error fluctuation using the Kalman filter is shown while in Fig. 22 the x/y position error fluctuation without Kalman filter.

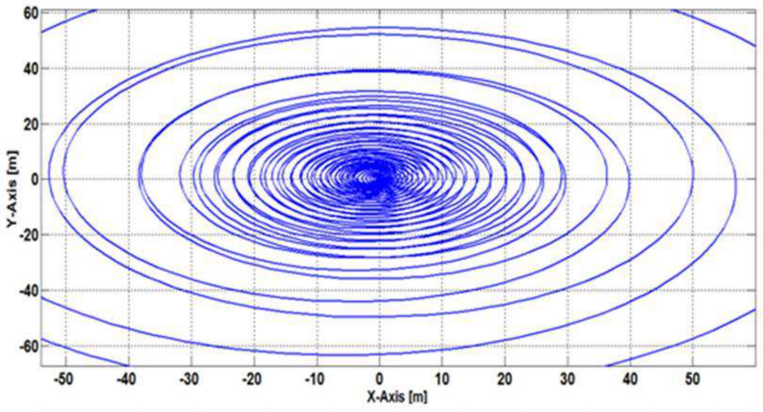
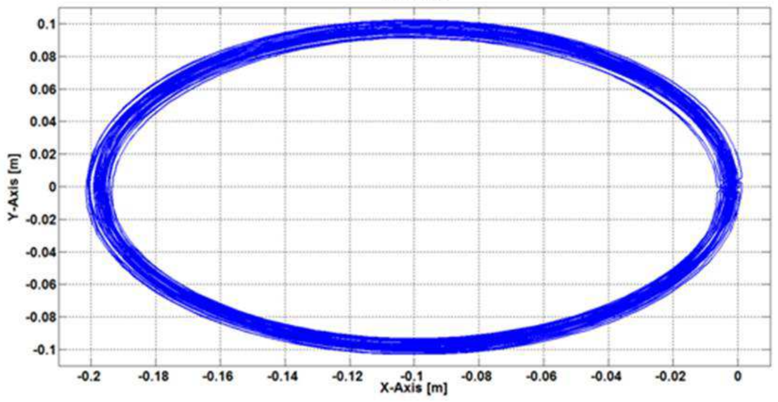



Fig. 20 - Trajectory reconstruction without (top) and with (bottom) Kalman filtering.

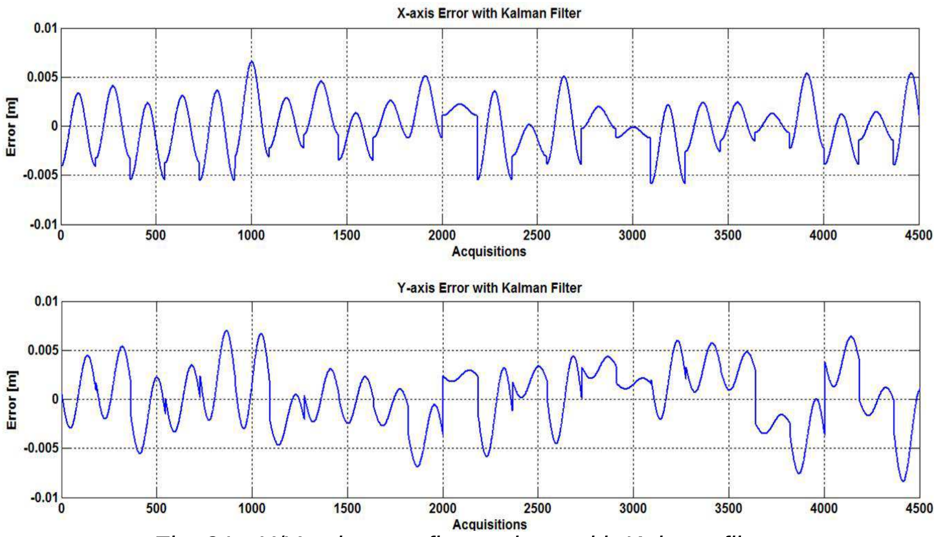


Fig. 21 - X/Y axis error fluctuations with Kalman filter.

Table 4 - Error distribution parameters (with Kalman filter)

Abs Max X-error	**0.0066m**
Abs Max Y-error	**0.0084m**
X-MSE (mean square error)	**3.30e-3m^2**
Y-MSE (mean square error)	**6.50e-3m^2**

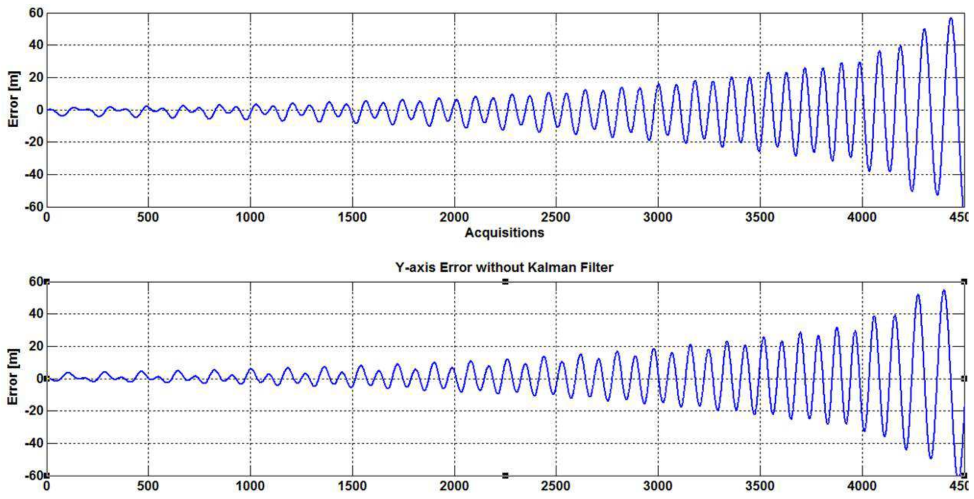


Fig. 22 - X/Y axis error fluctuations without Kalman filter.

Table 5 - Error distribution parameters (without Kalman filter)

Abs Max X-error	59.68m
Abs Max Y-error	60.57m
X-MSE (mean square error)	1.5256e+5m^2
Y-MSE (mean square error)	1.4943e+5m^2

2.8. Conclusions and future developments

A flexible and low-cost wireless GPS/Inertial system [24] which can be used for many kinds of applications is presented. The main features of prototype are low weight, high compactness, high autonomy, fast remote data managing and elaboration (Table 6). The future developments will be the GPS/Inertial data fusion, the replacement of MEMS sensor station with the new model which integrates a tri-axial magnetometer and an automatic thermal compensation, the replacement of the ZigBee module with the new model having a transmission range up to 1km and assembling all new modules and SMD components on the new PCB dual-layer board (Fig.23) to reduce more and more space and weight in order to increase system flexibility. In addition, the remote system GUI will be modified to manage data elaboration for various applications such as fall detection, body motion recognition, inertial navigation, etc. Many kind of tests in several scenarios will be conducted in order to demonstrate flexibility and general purpose capability of platform.

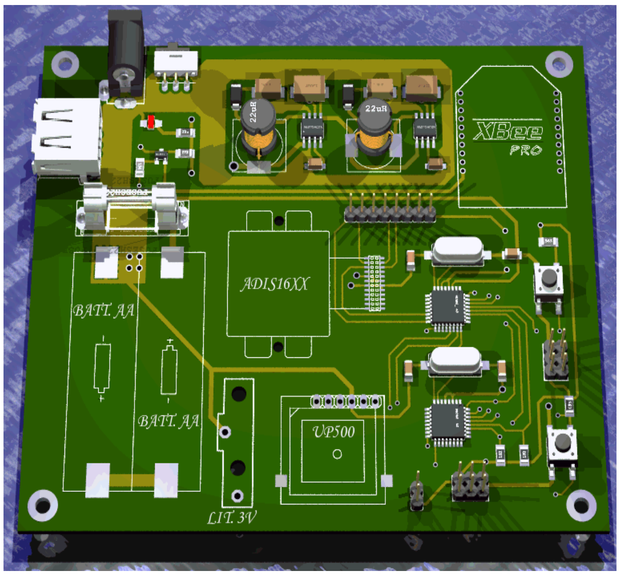


Fig. 23 - 3D PCB system board v2.3 image.

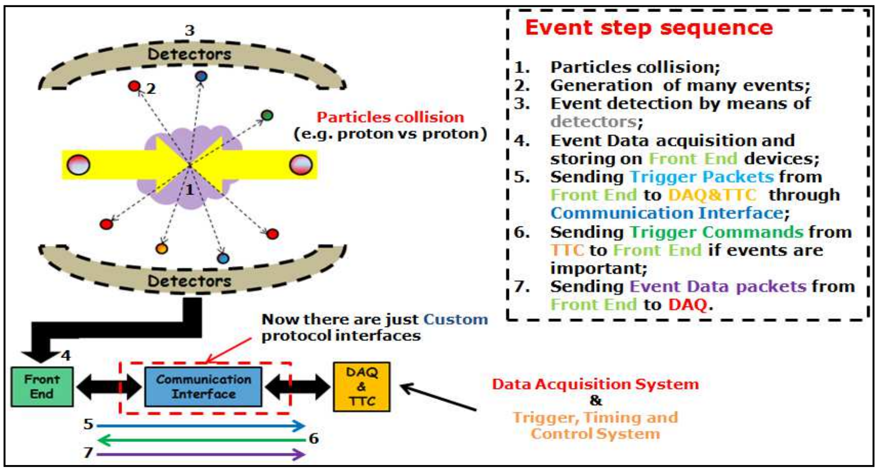
Table 6 - Main technical features

Dimensions [mm]	83x76x55
Weight [g]	210 (450 with camera)
Maximum transm. range [m]	80 (outdoor)
Inertial frame transm. rate [Hz]	100
GPS channels	32
GPS sensitivity (Track, Nav) [dBm]	-159
GPS frame transm. rate [Hz]	5
Position resolution with EGNOS [m]	< 5 (experimental)
Accmeters dynamic range [g]	±10
Accmeters sensitivity [mg]	2.522
Accmeters axis non-orthogonality [°]	± 0.25
Accmeters temp. coefficient [ppm/°C]	100
Gyros dynamic range [°/s]	±300, ±150, ±75
Gyros sensitivity [°/s]	0.07326, 0.03663, 0.01832
Gyros axis non-orthogonality [°]	± 0.05
Gyros temp. coefficient [ppm/°C]	600
Nominal input voltage [V]	2.4
Nominal input energy [Wh]	6.36
Max power consumption [mW]	625
Max battery autonomy [hr]	9 (working continuosly)

3. DEVELOPMENT OF AN EMULATION PLATFORM FOR THE FF-LYNX PROJECT

3.1. Introduction

Before describing the topic of this chapter, it is important to introduce the FF-LYNX (Fast and Flexible links) project. This project was promoted and financed by INFN (Istituto Nazionale di Fisica Nucleare), the most important research institute in Italy within the field of High Energy Physics (HEP) experiments. This project, started in January 2009, was born with the first aim to define a new serial communication protocol, to satisfy the typical requirements of HEP scenarios. It was intended to become a new flexible standard within different experiments minimizing development costs and efforts, because today each HEP experiment uses a different kind of custom communication protocol. The second aim of FF-LYNX project was to implement this protocol in radiation-tolerant, low power interfaces.
High Energy Physics (HEP), is a branch of physics that studies the most basic constituents of matter, i.e. subatomic particles, and their interactions. Particle accelerators are the main instruments for High Energy Physics. They are complex machines that produce beams of high energy particles. A typical HEP experiment consists in colliding particle beams and analyzing the results of the collisions using particle detectors around the interaction point. Large Hadron Collider (LHC) [25] at CERN (the European Organization for Nuclear Research) in Geneve (Switzerland) is the largest and most powerful particle accelerator over the world.
The electronic architectures used into experiments are very similar with respect to systems for acquisition of data from sensors and for control and management of the detector. Signals generated by sensors in particle detection are handled by Front-End (FE) electronics embedded in the detectors and transferred to remote data acquisition (DAQ) systems. In Fig. 24 a schematic representation of a common HEP experiment.



Event step sequence
1. Particles collision;
2. Generation of many events;
3. Event detection by means of detectors;
4. Event Data acquisition and storing on Front End devices;
5. Sending Trigger Packets from Front End to DAQ&TTC through Communication Interface;
6. Sending Trigger Commands from TTC to Front End if events are important;
7. Sending Event Data packets from Front End to DAQ.
3
Detectors
2
Particles collision
(e.g. proton vs proton)
1
Detectors
Now there are just Custom protocol interfaces
4
Front End
Communication Interface
DAQ & TTC
Data Acquisition System
&
Trigger, Timing and Control System
5
6
7

Fig. 24 - Schematic example of HEP experiment.

After a collision between two particles such as two protons, there are the generations of new different particles (collision events) which have to be captured by the detectors that surround the interaction point. These detectors, formed by many high performance sensors (pixels, strips, etc), are interfaced with Front-End (FE) devices that acquire signals and execute a signal conditioning (amplification, shaping, buffering, analog to digital conversion). So these raw digital data organized, in Event Data packets, are stored onto FE memory buffers temporarily. Since the interesting events are a very small fraction of the total, the total amount of data has to be filtered. To do this, a small amount of key information about collision event, in the guise of Trigger packets, is send to TTC (Trigger, Timing and Control) system which performs a fast, approximate calculation and identify if that event is significant or not. If the event is important, then a Trigger command is sent back to FE electronics to command a data readout. At this point, that Event Data packet is sent towards the DAQ (Data Acquisition System). In this way, the amount of data to be transferred is reduced to rates that can be handled by the readout system (a typical order of magnitude is hundreds of MB/s from each FE device), and only the interesting events are selected. Clearly each packet is sent through the communication channel based on different custom protocols.
The part of work presented in this chapter constituted two phases of the FF-LYNX design flow and was carried out at the Pisa division of INFN. It dealt with the simulation of FF-LYNX System-C interface models using a simulation platform and the design of an FPGA emulation platform for verify and test those interface models. In the first phase, these interfaces, defined in System-C language, are simulated using an ISE (Integrated Simulation Environment) platform. In the second phase of work, these interfaces were implemented on FPGA emulation platform to continue the verify process.

3.2. FF-LYNX protocol basis

The FF-LYNX protocol [26] is a double-wire serial protocol defined at the data-link layer of the ISO/OSI model. The two separate wires correspond to clock and data lines. It guarantees a high level of flexibility as regards data rate and data format. It is possible to communicate with three different data rates: 4xF, 8xF and 16xF, as shown in Fig. 25; F represents the frequency of the reference clock. Considering the LHC reference clock (40 MHz) there are 160, 320 and 640 Mbps respectively. The main feature of FF-LYNX protocol is the time multiplexing of two channels, named THS and FRM. The THS channel is used to transmit Triggers, Frame Headers and Synchronization patterns and employs two bits. The FRM channel is used to transmit data packets (information inserted into one or more data frames as Fig, 26) and employs 2, 6 or 14 bits in the three data rate options. A data packet is a high-level transmission unit which can be formed by several 16bit words. This data packet can fit a single data frame (if packet is formed by less than 16 words) or can be splitted into several data frames. It uses two kinds of data packets: the Variable Latency Frame (VLF) and the Fixed Latency Frame (FLF) packets, where the latency is defined as the data packet transfer time. The VLF is a generic data frame type while the FLF is used as trigger data frame type. The robustness of

critical information against transmission errors is obtained by means of Hamming codes and custom encoding techniques.
These techniques guarantee the correct recognition of commands and the reconstruction of their timing in the THS channel. In the FRM channel single bit-flips are corrected and burst errors are detected. There is a significant reduction of the number of physical links thanks to the use of the same protocol for the transmission of triggers, fixed and variable latency reducing the overall material budget. Hence, this protocol is suitable for the distribution both of DAQ signals and of Timing, Trigger and Control (TTC) signals, that is for the Up-Link (Front-End devices to DAQ system) and for the Down-Link (Trigger and Control System to Front-End devices) paths. On the THS channel, Triggers are higher priority signals with respect to Frame Headers and Synchronization commands; these latter can be transmitted only when there are no Triggers for at least three consecutive clock cycles, in agreement with the current specifications of the LHC experiments. On

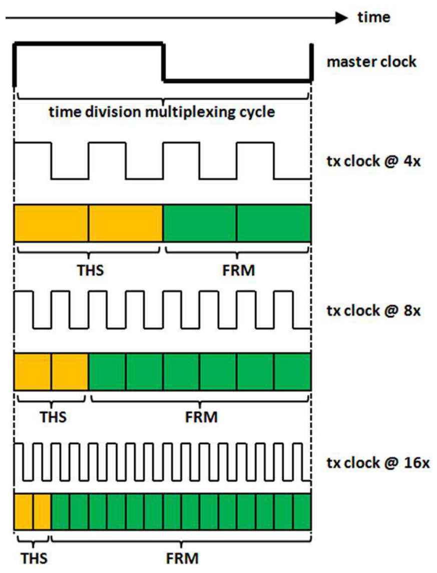


Fig. 25 - Channel partitioning through time-division multiplexing (TDM), with the master clock taken as the period for the TDM cycles.

FRM channel the Data Frames are tagged by Frame Headers transmitted on the THS channel.

The structure of a data frames is shown in Fig. 26. It is formed by: the Frame Descriptor (FD) which contains information such as the length of the frame and the type of data transmitted, the Label which represents a field that can be employed to add optional information, the Payload which constitutes the user data and a Cycle

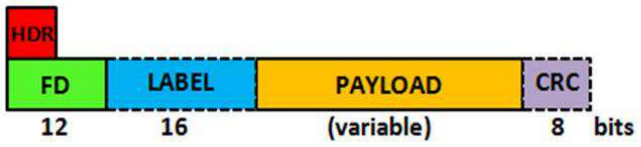


Fig. 26 - The basic FF-LYNX frame structure.

Redundancy Check (CRC) that can be optionally applied to the Payload to increase robustness against transmission errors.
In FF-LYNX system, as already mentioned, there are two categories of data with respect to latency constraints, the VLF and the FLF packets: the former have no latency constraints, while the latter must have a fixed latency.
The FF-LYNX protocol is implemented in Transmitter (TX) (Fig. 27.a) and Receiver (RX) (Fig. 27.b) interfaces with a serial port (DAT) on one side and two parallel ports (16-bit port for the VLF packets, 2/6/14-bits port for the FLF ones) with their control (data_valid, get_data, trg) and configuration signals (e.g.: flf_on, label_on) on the
other side. Control signals are used by host devices to manage the data

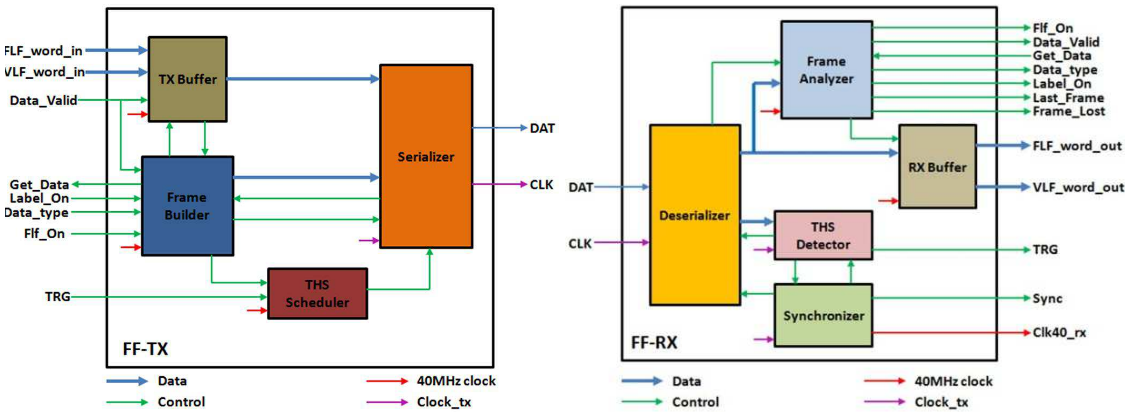


Fig. 27 - Functional architecture of the FF_TX (a) and FF_RX (b) interfaces.

transmission operations.

The FF-TX Transmitter is constituted of the following modules:

1. TX Buffer: it is structured as two FIFOs, for storing input data on VLF bus and on FLF bus.

2. Frame Builder: it controls the assembly of frames for the transmission of data stored in the FLF and VLF FIFOs.
3. THS Scheduler: it works out the arbitration between triggers and frame headers. It receives TRG and HDR commands and passes them to the Serializer avoiding THS sequences overlaps.
4. Serializer: It generates the serial output stream by receiving the Frame Descriptor field from the Frame Builder and frame words from the VLF/FLF FIFO. In addition, it sends TRG and HDR patterns into the THS channel, according to the commands that arrive from the THS Scheduler.

The FF-RX Receiver is constituted of the following modules:

1. Deserializer: It converts the FF-LYNX serial data stream into parallel form. it separates the THS channel and the FRM channel and provides the data words to store into the RX Buffer.
2. THS Detector: It detects the sequences of triggers, headers and synchronization patterns in the THS channel;
3. Synchronizer: It generates the reference clock on the base of information coming from the THS Detector.
4. Frame Analyzer: It controls the reception of data frames, their storing into the RX Buffer and the transmission of stored data to the receiver host.
5. RX Buffer: It buffers data to send to host devices in parallel form.

3.3. FF-LYNX top-down design flow

As already mentioned, the FF-LYNX project was to follow a well-defined top-down design flow (Fig. 28). This flow consists of six main phases: The protocol definition phase in which the communication interfaces are modeled using System-C language, the high-level validation phase which consists in verify the proper functionality of protocol interfaces employing a simulation platform called ISE (Integrated Simulation Environment), the definition of hardware interfaces using VHDL language, the implementation of interfaces on FPGA devices for the emulation phase and at the end, the design of test ASIC chip implementing these FF-LYNX TX/RX interfaces. In this chapter, mainly the FPGA prototyping phase will be described in detail.

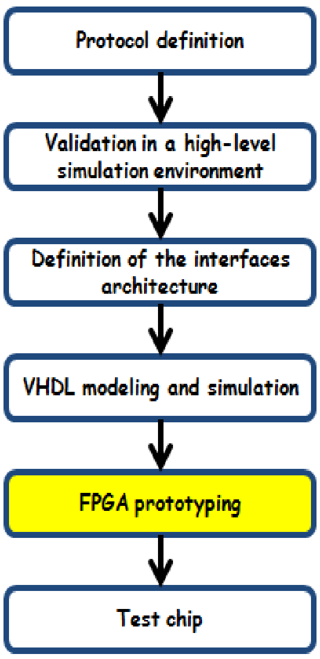

Protocol definition
Validation in a high-level simulation environment
Definition of the interfaces architecture
VHDL modeling and simulation
FPGA prototyping
Test chip

Fig. 28 - The FF-LYNX design flow.

3.4. FF-LYNX protocol system-C modeling

3.4.1. The ISE (Integrated Simulation Environment) platform

After the theoretical definition of protocol structure, an ISE (Integrated Simulation Environment) platform, based on models written in System-C language [27], was developed. These System-C models describe protocol interfaces, electrical links and I/O test modules with a timing accuracy at clock cycle level. The aim of the ISE platform is to simulate and characterize readout architectures based on FF-LYNX communication protocol, in this case study, with input data compatible with possible working environment in which this protocol could be employed. For this goal, physics GEANT4 input data are used and FoM (Figures of Merit), defined in Table 7, are evaluated.
With this ISE platform it is possible to conduct all kind of analysis in different operating conditions, setting different values of link speed, trigger rate, packet rate, packet average size, bit error rate in electrical serial links. It is possible also to include injection of errors in communication links and memory blocks. All the models that form the ISE are known as "Simulator".
The developed System-C link simulator is composed by two main modules: the FF-LYNX TX interface and the FF-LYNX RX interface. This model architecture is parameterized and modular, allowing the reusability of System-C code and the run-time behavior tuning. This feature is important during the simulation phase when frequent changes in parameter values are needed for FoM estimations.
The Simulator is laid out in a Client/Server architecture (Fig. 29). In the Server side there are two main blocks, the Test Bench and the Server Main modules, while in the Client side there are the Sim Framework and the Client Main modules.

Table 7 - Figures of Merit (FoM)

Figures of Merit (FoM)	**Description**
Sent Pkt	number of VLF packets sent
Lost Pkt	number of VLF packets lost
LPR *(Lost Packet Rate)*	Lost Pkt over Sent Pkt
CPDR *(Corrupt Packet Descriptor Rate)*	number of VLF packets received with incorrect length
CPPR *(Corrupt Packet Payload Rate)*	number of VLF packets received with damaged payload
Mean, Min, Max PL *(Mean, Min, Max Packet Latency)*	mean, min and max value of the packet latency
Std PL *(Standard Deviation of Packet Latency)*	standard deviation of the packet latency
Sent Trg *(Sent Triggers)*	number of triggers sent
Lost Trg *(Lost Triggers)*	number of triggers lost
Lost Hit	number of FLF packets lost or corrupted
LTR *(Lost Trigger Rate)*	lost triggers over sent triggers
FTR *(Fake Trigger Rate)*	the rate at which fake triggers are received
Lost Hit Rate	lost hits over sent hits

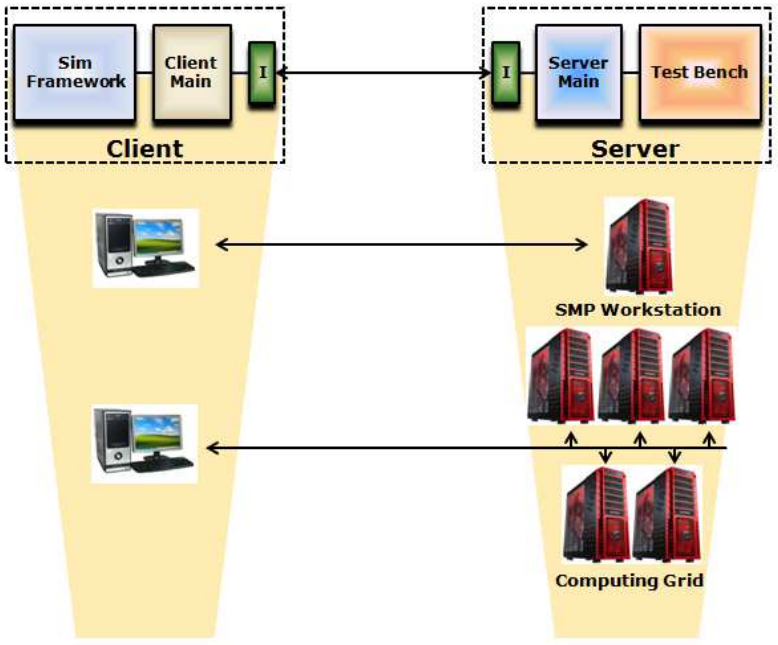


Fig. 29 - The ISE Client-Server architecture; it can be implemented on SMP (symmetric multi processor) workstation or computing grid.

Concerning the Server side, the Test Bench module contains the System-C protocol interfaces and its task is to transfer input data from the Server Main to the protocol interfaces and then to receive protocol interface outputs. The Server Main behaves as a functional master module, it stores temporarily both data coming from the Client and data waiting to be transmitted back to it. Both the Server and the Client have a Sim Interface module that interfaces the Server side with the

Client side. The message passing is implemented on top of TCP/IP sockets. As regards the Client side, the Sim Framework module is made of a Stim_Gen module that generates the stimuli patterns and a FoM_Gauge module that gauges the figures of merit from the simulation results. The Client Main manages their initialization and provides the highway through which data flows from the Sim Interface module to the Sim Framework module and viceversa.
The ISE architecture, being modular, allows to easily change every single module of the system, as long as the module interface remains the same. Thanks to the Client/Server approach there is a high degree of flexibility since the Server side can be relocated on a different (remote) machine or re-implemented for another architecture type (i.e.: FPGA emulator), without modifying the Client side. In this environment a typical simulation is based on one or many "runs" which can have different simulator configurations (parameter settings) to evaluate how the system behaves after these variations. As shown in Fig. 29, the ISE platform can be also implemented both in Symmetric Multi-Processing (SMP) machines (i.e.: multi-core and/or multi-processor) and in powerful computing grids (i.e. hundreds or thousands of processors) by spreading the load on multiple processing units, in order to decrease the simulation time and conduct longer and deeper analysis.
An example of a simulation carried out in the ISE environment is shown in Fig. 30 where the packet latency time (mean, max, min and standard deviation metrics) related to VLF data packets varies with the protocol speed (4x, 8x, 16x). For this analysis the physical layer considered is a coaxial cable. In Fig. 31 and 32 there are two examples of performance analysis that can be carried out, since the early protocol development steps, by using the ISE platform.
The analysis in Fig. 31 and 32 regards the evaluation of mean sync time and false sync percentage for different values of the N_unlock and of the N_lock thresholds used in the Synchronization module of the FF-RX interface. Sync time, expressed in 40 MHz reference clock cycles, is the time for system re-synchronization after a synchronization loss, while false sync percentage depends on fake synchronization events. N_unlock and N_lock are thresholds that indicate the minimum number of detected synchronization sequences on one of the possible THS channels (4, 8 or 16 in the three speed options) for synchronization unlocking and locking respectively. The synchronization mechanism is based on the counting of THS sequences in each channel and the reaching of two counting thresholds (a high threshold, N_lock, and a low one, N_unlock) was chosen to distinguish a synchronization lock state (when synchronization is considered as acquired) and an out-of-lock state (when synchronization is being looked for).

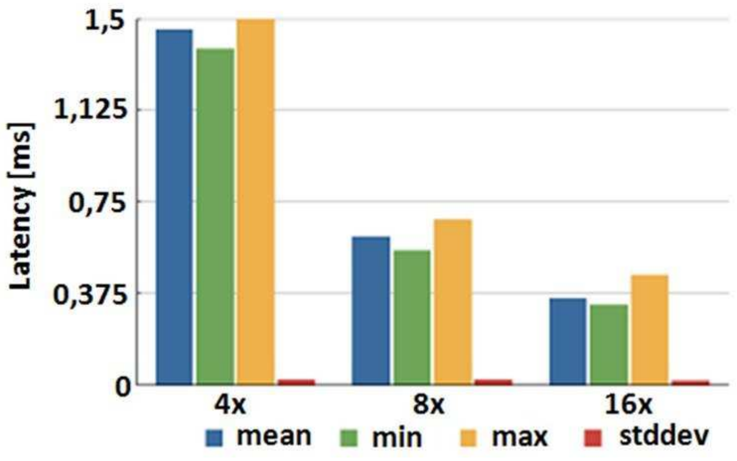


Fig. 30 - Mean, minimum, maximum and standard deviation of packet latency with different link speeds.

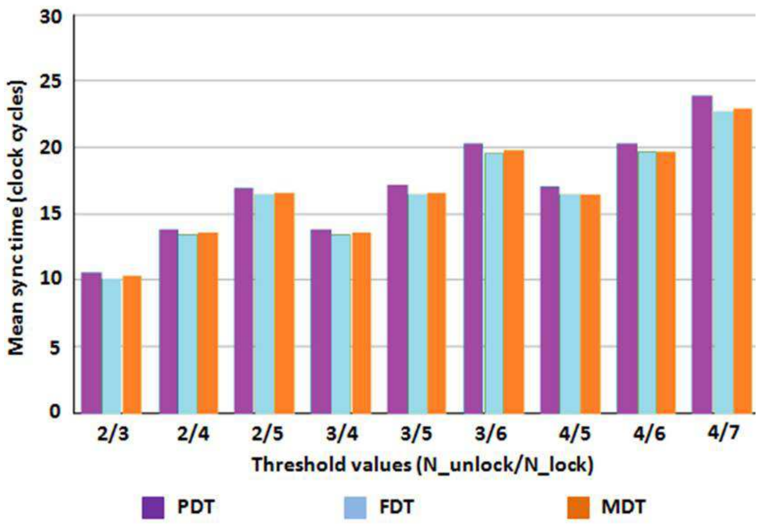


Fig. 31 - Mean synchronization time for different threshold settings.

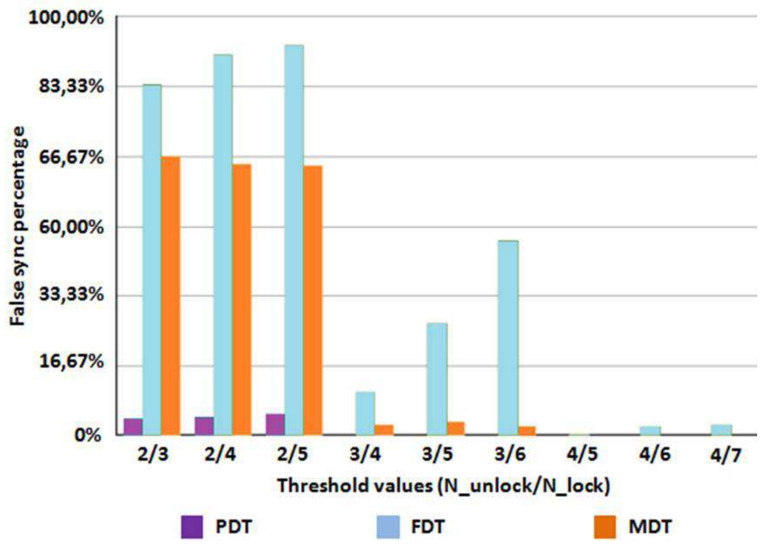


Fig. 32 - False synchronization percentage time for different threshold settings.

This mechanism is hence called Dual Threshold (DT): the transition from the out-of-lock state to the sync lock state takes place when one of the counters reaches the high threshold (thus becoming the in-charge counter) while the inverse transition occurs when a counter that is not in charge reaches the low threshold. As already mentioned, the Synchronization module is used for the detection of THS channel and the recovery of the reference clock. In these examples three different synchronization algorithms are considered: Privileged Dual Threshold (PDT), Fair Dual Threshold (FDT) and Mixed Dual Threshold (MDT). Using the PDT algorithm, when a counter hits the high threshold, it resets all the other counters but not itself. With the FDT algorithm, as soon as a counter hits the higher threshold, it resets all the counters including itself. The MDT algorithm combines the two previous variations giving an intermediate level of privilege to the in-charge counter. Taking into account the results of these system-level simulations the PDT synchronization algorithm was used for further development steps of the protocol and hardware implementation of its building blocks. Indeed, with N_unlock and N_lock equal to respectively 3 and 4, the PDT synchronization technique represents the best trade-off between mean sync time and false sync percentage.
In general terms, the ISE platform allows to tune the parameters of a generic system architecture to design (FF-LYNX protocol interfaces, in this case) according to results of performance analysis and to analyze in detail the behavior of the system during a typical working. In ISE the system architecture to verify is defined using System-C models. If the system works properly during simulation then, on the basis of System-C models and of performance analysis, HDL models are defined and implemented onto FPGA. Thanks to the flexibility of ISE platform, it is possible to use the simulator in the FPGA emulator by replacing the server section with the emulator without modifying the client section. As already said previously, this client-server architecture allows to take advantage of socket communication

and to provide a scalable environment suitable for Symmetric Multi-Processing (SMP) workstations or computing grids.

3.4.2. The ISE applications

The developed ISE environment can be also used for an early evaluation of the impact of a new communication protocol and its hardware solutions on the performance of data acquisition systems; this drives to great advantages in terms of reduction of development time and costs for a new project/experiment or an upgrade of an ongoing one. Particularly, the ISE environment was used to characterize the Track-Trigger architecture proposed for the upgrade of the CMS Silicon Tracker, whose data are used in CMS to reconstruct the trajectories of the charged particles. Achievable data-rate was evaluated comparing different algorithms used for clustering, pairing and track-let finding. The performance of the readout system (e.g.: lost hit rate, corrupted hit rate, cluster rate, pair rate) was analyzed using different protocol configurations (e.g.: size of FLF packets) and hardware implementations (e.g.: Front-End buffer size, number of links, link speed). In performance analysis physics event classes (i.e.: loss rate, corrupted hit rate, cluster rate, pair rate for given physics events) were considered.
By using GEANT4 data as ISE inputs an analysis was conducted; examples of the achievable results are reported in Fig. 33 and 34 if using the FF-LYNX protocol described in Section 2 for the upgrade of the CMS tracker. Particularly, Fig. 33 shows the achievable data rate, Mbps, at different Z values (longitudinal positions in the CMS tracker) (Z0=0 cm, Z1=69 cm, Z2=137 cm, Z3= 206 cm) and for different detector layers while Fig. 34 reports the achievable transmission efficiency as a function of the effective available bandwidth.

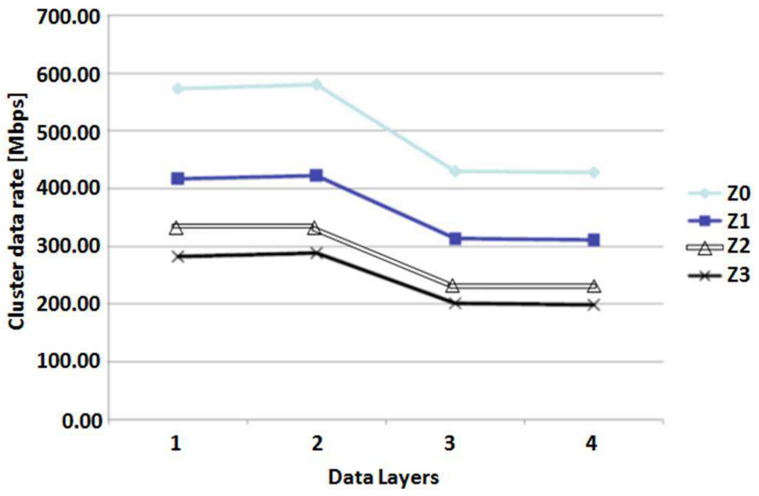


Fig. 33 - Cluster data rate (Mbps) at different Z and different detector layers.

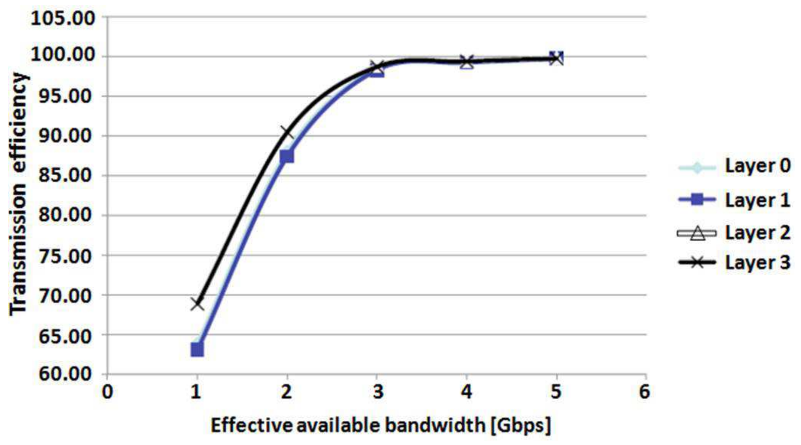


Fig. 34 - Transmission efficiency vs. available bandwidth (Gbps) for Z=Z0, different detector layers.

3.5. FPGA-based emulation platform

3.5.1. Emulation system overview

In order to test the FF-LYNX protocol interfaces and to validate the ISE simulation results, an FPGA-based emulator platform was designed. This emulator system (Fig. 35) is based on a C++ Graphical User Interface (GUI), used for the configuration and the control of the emulator, and on the VHDL emulator core. The VHDL emulator is implemented on an Altera Stratix II GX FPGA device (EP2SGX130GF1508 which provides 133,000 equivalent Logic Elements, 7 Mbit total RAM memory, 8 PLLs, 78 LVDS channels, 63 multipliers and adders blocks and 20 Transceiver channels with a data rate from 600 Mbps to 6.375 Gbps) [28] housed on a commercial PCIe board (PLDA XpressGXII board) [29] which can handle a 3.6 GB/s data rate (effective, full duplex). This FPGA development board was chosen to guarantee a large bandwidth to perform extensive emulations with high data rates with an acceptable efficiency in terms of emulation time. A typical real working scenario is to receive data from a CMS Tracker Front-End chip with a 1.8 Gbps data rate (from physics simulation). Three or four FF-LYNX TX/RX interfaces working at 640 Mbps (maximum data rate allowed by the FF-LYNX protocol with a reference clock frequency of 40 MHz) could handle this data rate. The PCIe board, with its large bandwidth, allows realistic emulations. In our case, for one second of emulation, sending the test vectors and receiving the emulation results take about 62.5ms. In addition, having a single FPGA development board which can be mounted on a workstation instead of having a platform composed by many separated parts, it allows to increase the level of compactness of the whole emulation system. This is a key feature towards the reuse of the same emulator environment as test bed when doing irradiation tests on the ASIC chips implementing rad-tolerant FF-LYNX building blocks.

Fig. 35 - Emulation system: GUI, Workstation and Development board.

3.5.2. V HDL emulation system

The VHDL emulator system is formed by three main functional blocks, as shown in Fig. 36: the PCIe XpressLite core, the Interface Logic and the FF-Emulator core.

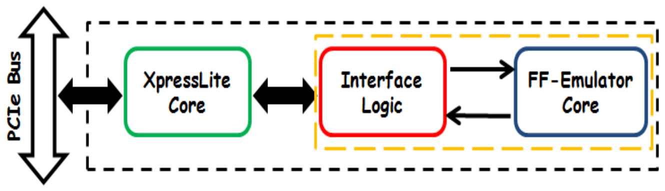


Fig. 36 - VHDL emulator system.

The PCIe XpressLite core in Fig. 36 is an IP-core block provided by PLDA. It manages the communication with the PCIe bus and therefore with the host workstation. The Interface Logic is another block provided by PLDA, but it can be customized by the user and it is used to interface the PCIe XpressLite core to the user custom application block that is the FF-Emulator core. The FF-Emulator is a custom developed core whose block diagram is shown in Fig. 37.
The FF-emulator core is divided in two modules: a Test Controller (TC) that manages the emulation test and the FF-LYNX TX and RX interfaces that represent the Device Under Test (DUT). The TC has a simple structure based on the following building blocks:
TC core that includes two functional paths. The TX path manages the data transmission toward the FF-TX block. It is made of the TX Controller and the TX RAMs. The RX path manages the data reception from the FF-RX block. It is made of the RX Controller and the RX RAMs;
TC configuration registers that contain test parameters as emulation window duration, number of VLF packets, presence or not of FLF packets, size of the FLF packets, TX TC buffer size, etc;
The TX controller starts the transmission of a trigger (or of a FLF packets, when they are enabled) or the transmission of a VLF packet at pre-defined clock cycles (TX Time Stamps). The arrival timing of the received triggers and frames is stored and then compared with the TX to evaluate the trigger and the packet latency. The TX and RX RAMs are divided in five different types of RAM:

VLF_TS (Variable Latency Frame Time Stamp) RAM to store the time stamps (TS) associated to VLF data packets transmitted/received to/from FF-LYNX interfaces;
VLF_LEN (Variable Latency Frame Length) RAM to storie the length (number of 16bit words) of the VLF data packets transmitted/received to/from FF-LYNX interfaces;
VLF_DW (Variable Latency Frame Data Word) RAM to store the VLF data packets transmitted/received to/from FF-LYNX interfaces;
TRG_TS (Trigger Time Stamp) RAM to store the time stamps associated to triggers or FLF data packets (if they are enabled) transmitted/received to/from FF-LYNX interfaces;
FLF_DW (Fixed Latency Frame Data Word) RAM to store the FLF data packets (if they are enabled) transmitted/received to/from FF-LYNX interfaces.
The TX and RX Controllers are made up of (i) time stamp counters whose current values are compared with values stored in the TX TS RAMs or stored in the RX TS RAMs, (ii) FSMs (Finite State Machine) to control the test flow and (iii) two FIFO buffers (only for TX Controller): one configurable buffer to store VLF data temporarily and another to store VLF data lengths associated to VLF data buffered on the first buffer. These two buffers allow to load a complete packet before sending it on FF-LYNX link.

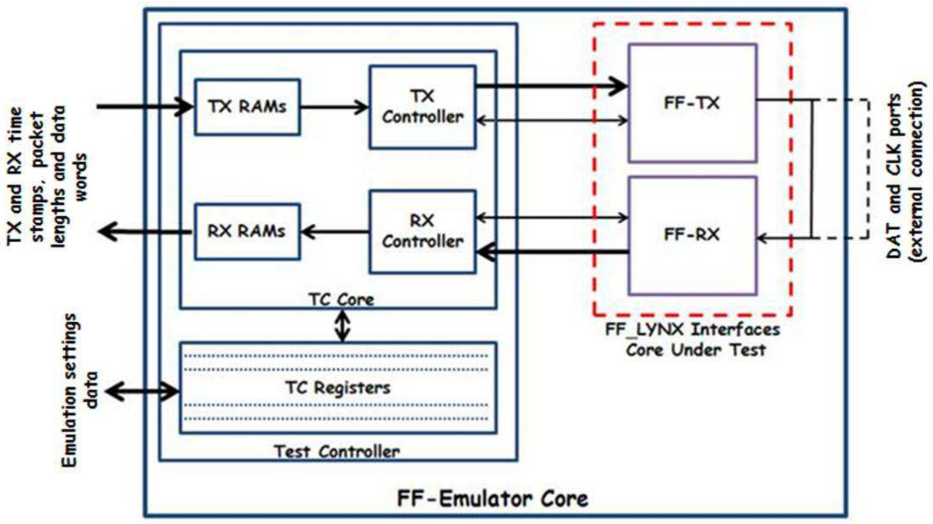

TX and RX time stamps, packet lengths and data words
TX RAMs
TX Controller
RX RAMs
RX Controller
TC Core
TC Registers
Test Controller
Emulation settings data
FF-TX
FF-RX
FF_LYNX Interfaces Core Under Test
DAT and CLK ports (external connection)
FF-Emulator Core

Fig. 37 - FF-Emulator core block diagram.

3.5.3. Emulator GUI software

The C++ GUI software is organized as shown in Fig. 38
It is based on three main parts: the PLDA drivers to interface the Host-PC with the PCIe PLDA board, the Data Manager to manage the emulator operations and the GUI (Fig. 39) to set all the emulation variables and to display emulation results. In the Data Manager block three important parts can be found: the EmuRun part which manages an emulation run on the base of several parameters set by user (as FF-LYNX interface data rate, emulation window duration, packet and trigger rate, number of triggers and packets, etc); the StimGen part that generates the VLF/FLF data packets and the time stamps and store them in the TX RAMs; the Figures of Merit (FoM) Gauge that, according to emulation results, allows to define some FoM (as shown in Table 7), useful for validating and verifying FF-LYNX protocol efficiency.

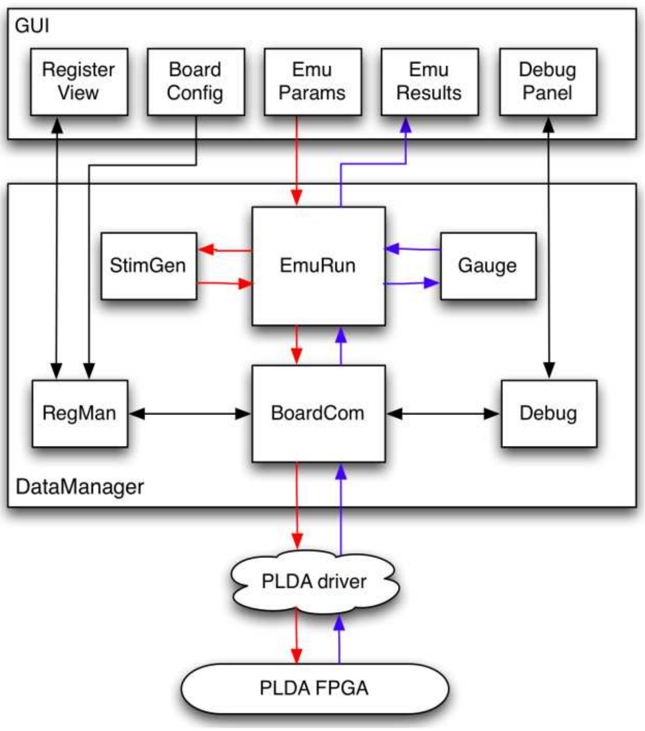
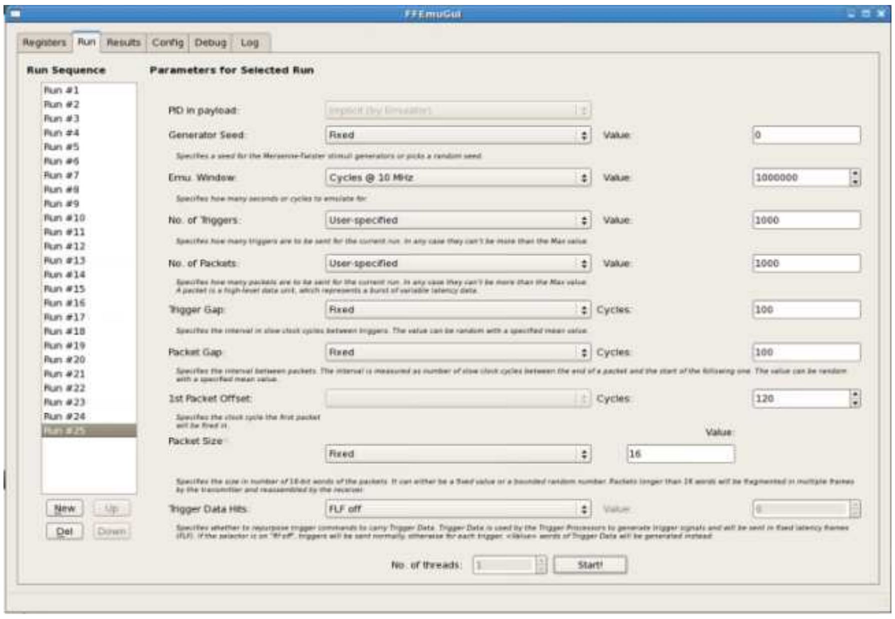


Fig. 38 - C++ emulator software block diagram.



Run #1
Run #2
Run #3
Run #4
Run #5
Run #6
Run #7
Run #8
Run #9
Run #10
Run #11
Run #12
Run #13
Run #14
Run #15
Run #16
Run #17
Run #18
Run #19
Run #20
Run #21
Run #22
Run #23
Run #24
Run #25

PID in payload: Implicit (by Emulator)

Generator Seed: Fixed Value: 0

Specifies a seed for the Mersenne-Twister stimuli generators or picks a random seed

Emu. Window: Cycles @ 10 MHz Value: 1000000

Specifies how many seconds or cycles to emulate for

No. of Triggers: User-specified Value: 1000

Specifies how many triggers are to be sent for the current run. In any case they can't be more than the Max value

No. of Packets: User-specified Value: 1000

Specifies how many packets are to be sent for the current run. In any case they can't be more than the Max value. A packet is a high-level data unit, which represents a burst of variable latency data.

Trigger Gap: Fixed Cycles: 100

Specifies the interval in slow clock cycles between triggers. The value can be random with a specified mean value.

Packet Gap: Fixed Cycles: 100

Specifies the interval between packets. The interval is measured as number of slow clock cycles between the end of a packet and the start of the following one. The value can be random with a specified mean value.

1st Packet Offset: Cycles: 120

Specifies the clock cycle the first packet will be fired in

Packet Size: Fixed Value: 16

Specifies the size in number of 16-bit words of the packets. It can either be a fixed value or a bounded random number. Packets longer than 16 words will be fragmented in multiple frames by the transmitter and reassembled by the receiver.

Trigger Data Hits: FLF off Value: 0

Specifies whether to repurpose trigger commands to carry Trigger Data. Trigger Data is used by the Trigger Processors to generate trigger signals and will be sent in fixed latency frames (FLF). If the selector is on "ff off", triggers will be sent normally, otherwise for each trigger, <Value> words of Trigger Data will be generated instead.

No. of threads: 1

Fig. 39 - Emulator GUI.

3.5.4. FF-LYNX emulator working

An emulation run is initialized by the user by means of the Emu GUI software. At the beginning it is necessary to set the FF-LYNX interface data rate (4x, 8x, 16x), the test configuration (choosing an internal or external connection between TX and RX modules under test), the TC TX buffer size, the emulation window duration, the number of packets and triggers, the size of VLF packets, the presence or not of FLF packets and the packet and trigger rate. Then the run can be started (Fig. 40). On VHDL emulator core side, by means of the Interface Logic module, the user emulation settings are stored on TC registers and the time stamps, the packet lengths and the data words, sent from the host PC, are stored into the TX RAMs. The TS counter of the TX Controller is started. When its value is equal to the value stored in the currently pointed locations of the VLF_TS or FLF_TS RAMs (as explained in Fig. 41), the system is ready to send triggers or packets towards the FF-LYNX TX interface. When the count value equals the TS RAM values, the TS and VLF_LEN RAM current pointers are increased by one. If count value is equal to VLF_TS RAM value then a number N of words of VLF_DW RAM are loaded and after stored on VLF data FIFO buffer. N is equal to the corresponding pointed value of VLF_LEN RAM and then it is stored on VLF data length buffer. After data storing into buffer, it needs to start interface operations. Data_Valid signal of FF-LYNX TX Back-end Interface is set high to show there are data packets ready for sending. If Get_Data signal of FF-LYNX TX Back-end Interface is low, data packets will wait for being sent, otherwise they are sent to FF-LYNX TX interface respecting the number N of VLF data words indicated by value stored on VLF data length buffer. If count value is, instead, equal to the current pointed value of TRG_TS RAM and if FLF packets setting is active (from GUI software) (as in Fig. 42), then the corresponding value of FLF_DW RAM is sent to the FF-LYNX TX interface. FLF packets are sent without waiting or buffering because FLF packets, having to maintain fixed latency, have priority over VLF packets. Then, Trigger signal of FF-LYNX TX Back-end Interface is set high. If FLF packets is not active, only Trigger signal is set high. All operations are executed until TS counter reaches the last value of TS RAMs. After the data transmission operations towards the FF-LYNX TX interface block is finished, FF-LYNX TX interface block will send data to FF-LYNX RX interface block according to the FF-LYNX protocol. The RX Controller in the TC core will manage the interface with the FF-LYNX RX Back-end plus the data reception and storage tasks on the respective RX RAMs. At the end of this loop-fashion process, the RX RAM is read by the host system which, as already explained, will use it for generating FoM.

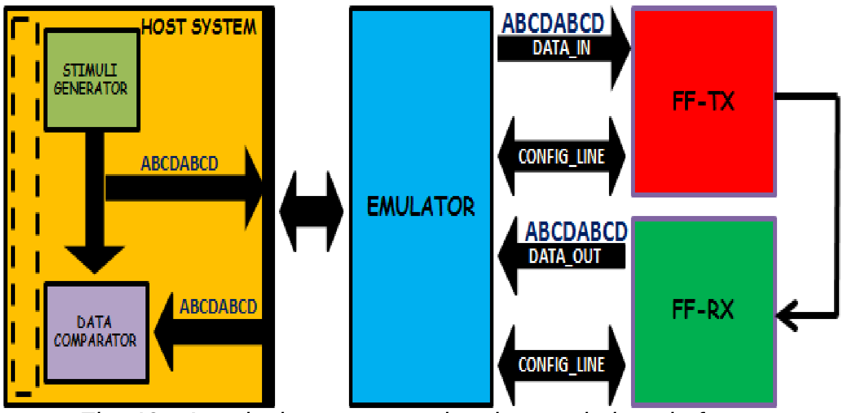


Fig. 40 - A typical test setup using the emulation platform.

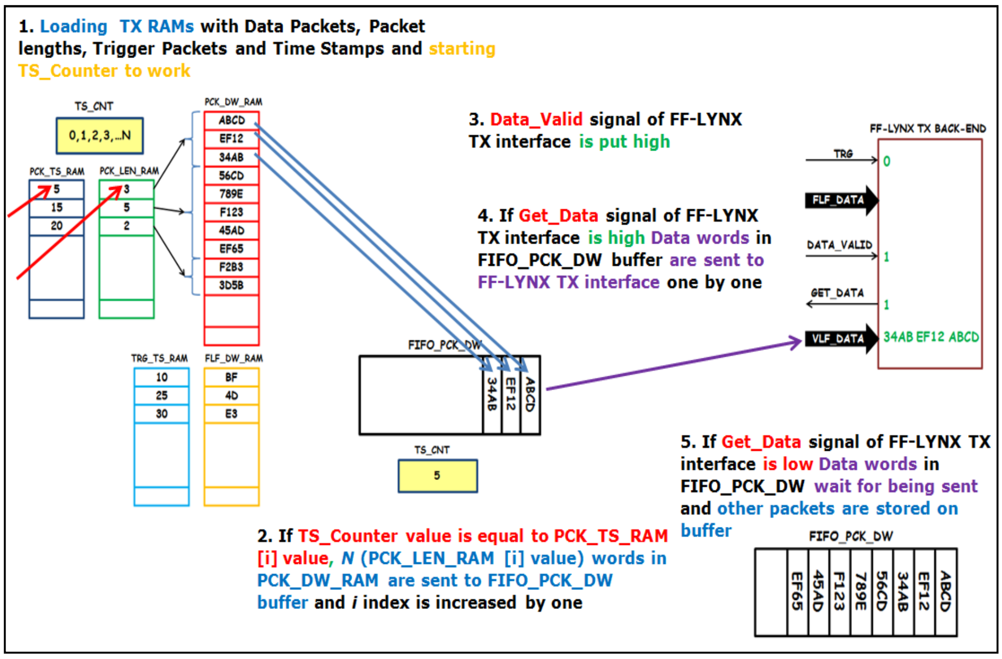

1. Loading TX RAMs with Data Packets, Packet lengths, Trigger Packets and Time Stamps and starting TS_Counter to work
TS_CNT
0,1,2,3,...N
PCK_TS_RAM
5
15
20
PCK_LEN_RAM
3
5
2
PCK_DW_RAM
ABCD
EF12
34AB
56CD
789E
F123
45AD
EF65
F2B3
3D5B
TRG_TS_RAM
10
25
30
FLF_DW_RAM
BF
4D
E3
3. Data_Valid signal of FF-LYNX TX interface is put high
4. If Get_Data signal of FF-LYNX TX interface is high Data words in FIFO_PCK_DW buffer are sent to FF-LYNX TX interface one by one
FF-LYNX TX BACK-END
TRG
0
FLF_DATA
DATA_VALID
1
GET_DATA
1
VLF_DATA
34AB EF12 ABCD
FIFO_PCK_DW
34AB
EF12
ABCD
TS_CNT
5
5. If Get_Data signal of FF-LYNX TX interface is low Data words in FIFO_PCK_DW wait for being sent and other packets are stored on buffer
FIFO_PCK_DW
EF65
45AD
F123
789E
56CD
34AB
EF12
ABCD
2. If TS_Counter value is equal to PCK_TS_RAM [i] value, N (PCK_LEN_RAM [i] value) words in PCK_DW_RAM are sent to FIFO_PCK_DW buffer and i index is increased by one

Fig. 41 - FF-LYNX emulator working (example of VLF packet transmission).

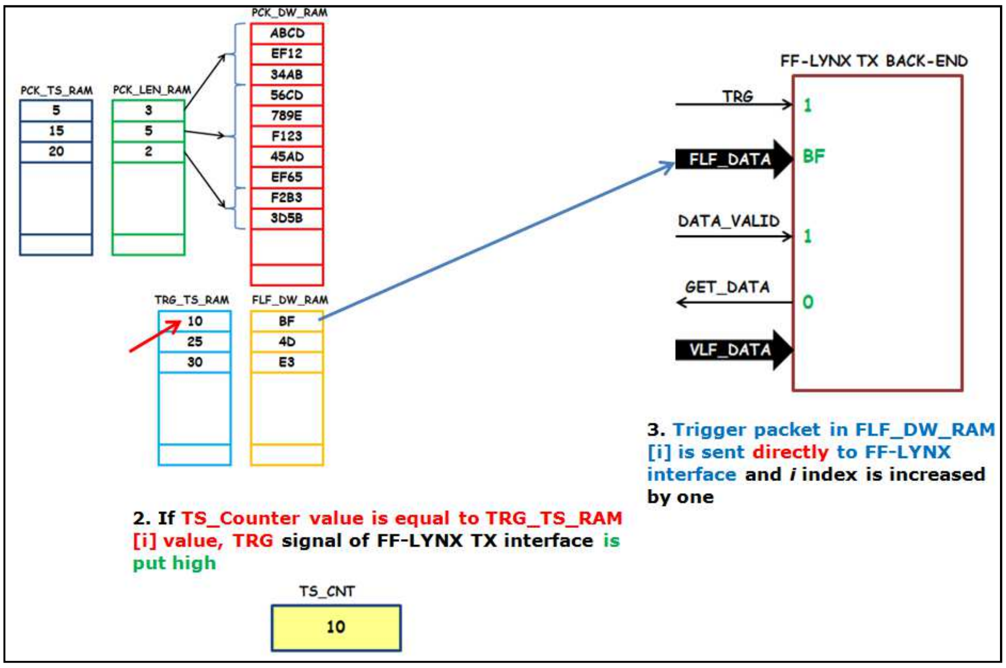

PCK_TS_RAM
5
15
20
PCK_LEN_RAM
3
5
2
PCK_DW_RAM
ABCD
EF12
34AB
56CD
789E
F123
45AD
EF65
F2B3
3D5B
TRG_TS_RAM
10
25
30
FLF_DW_RAM
BF
4D
E3
FF-LYNX TX BACK-END
TRG
1
FLF_DATA
BF
DATA_VALID
1
GET_DATA
0
VLF_DATA
3. Trigger packet in FLF_DW_RAM [i] is sent directly to FF-LYNX interface and i index is increased by one
2. If TS_Counter value is equal to TRG_TS_RAM [i] value, TRG signal of FF-LYNX TX interface is put high
TS_CNT
10

Fig. 42 - LYNX emulator working (example of FLF packet transmission).

3.6. Emulation test

Using the proposed emulation environment it is possible to extract some interesting graphs about the obtained FoM, thus assessing the performances of the communication protocol and its hardware building blocks, and evaluating their suitability when applied to the physics experiment of interest.
As example, next figures show some tests carried out using 8x FF-LYNX interfaces and links. Particularly, Fig. 43 shows the achieved packet latency, in 40 Mhz reference clock cycles, as a function of the packet size expressed in data words. The achieved results provide important suggestions about the most suited way to configure the communication protocol and its interfaces for the project of interest (e.g. an HEP experiment); indeed Fig. 43 shows that when the packet size increases over 52 data words there is a sharp increase of the packet latency. This increasing is due to packet queuing in TX buffer of FF-LYNX interface under test. Below 52 data words there is a proportional increasing, with an "heuristic law" of roughly 4 clock cycles of latency increase for each new data word in the packet size, simply due to the necessary time to send larger and larger packets.
Fig. 44 shows the Packet latency, in 40 MHz reference clock cycles, as a function of the trigger gap, i.e. the distance, still expressed in clock cycles, between consecutive triggers. Fig. 44 shows that when the trigger rate increases the packet latency increases too. Packet latency increases because trigger priority is higher than data priority and so, when increasing the trigger rate, VLF packets are forced to wait into buffer before being sent.
To be noted that in Fig. 43 and 44 for the packet latency all 3 main cost metrics defined in Table 7 were measured by emulation: min, max and mean PL.

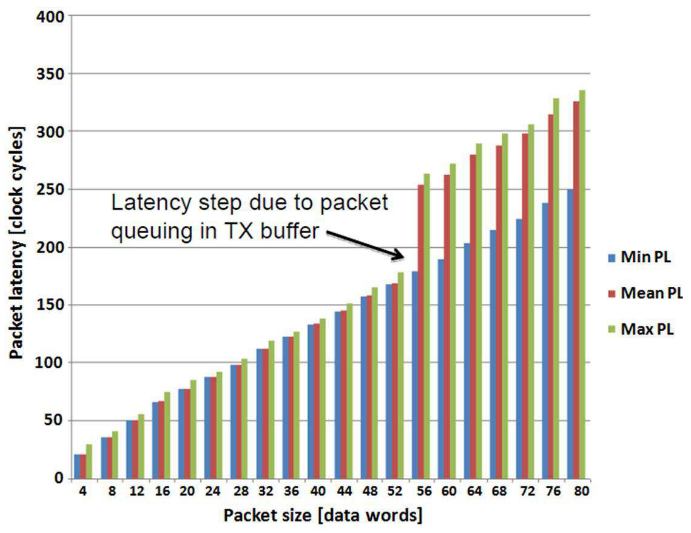


Fig. 43 - Packet latency vs. Packet size.

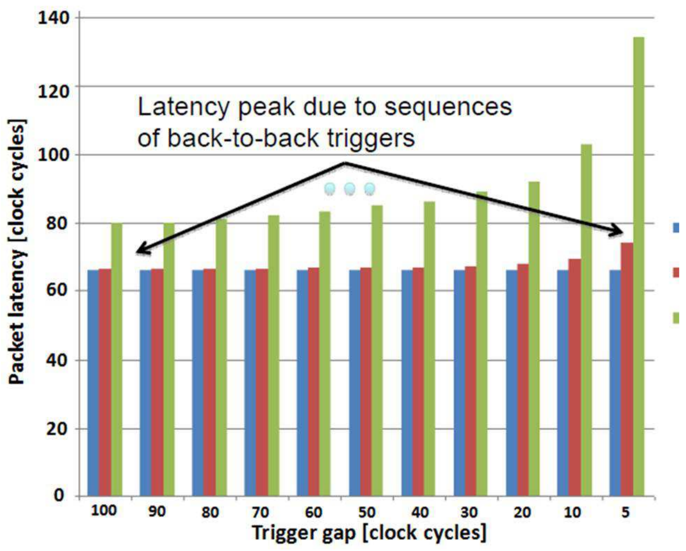


Fig. 44 - Packet latency vs. Trigger gap.

Fig. 45 shows the loss rate of data words as a function of the TC buffer size. The achieved results prove that by increasing the TC buffer size it is possible to decrease data word loss, because of a more data buffering. In Fig. 46 there is an important result that underlines the importance of emulator approach as a necessary part of protocol interface development and validation. It shows the differences between high level simulation and emulation when measuring, as example, the packet latency as a function of the distance (in 40 MHz reference clock cycles) of consecutive packet transmissions. As reported in Fig. 46, using a packet size equal to 8 words (16bit), when packet gap decreases at 30 clock cycles or lower there is a mismatch between simulation results obtained thanks to ISE platform and emulation results obtained with FPGA emulator. The difference is due to protocol modeling inaccuracies proper of the high-level abstraction approach. In the specific case of the FF-LYNX simulation environment the System-C model of the protocol interfaces does not fit perfectly the real physical protocol interfaces: the model does not consider the finiteness of TX buffer and a consequent possible buffer overflow with packet loss. These packets which would be really lost, instead, are seen as packets still buffered and so there is a fake packet latency increasing in Fig. 46 for the high level simulation vs. the real-world emulation.

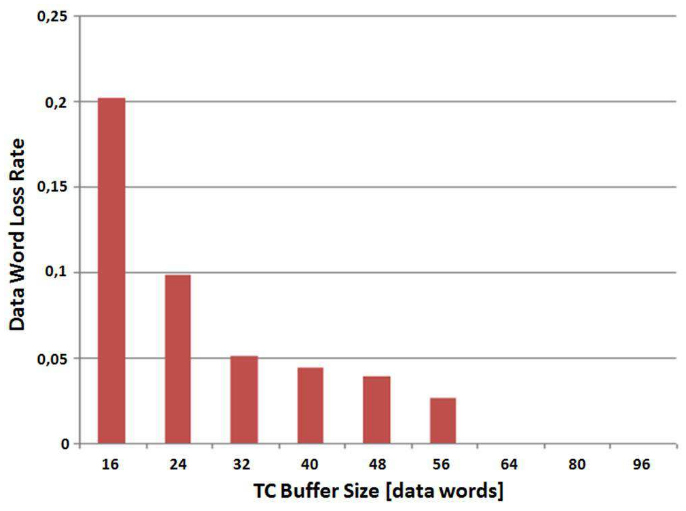


Fig. 45 - TC buffer size vs. data word loss rate.

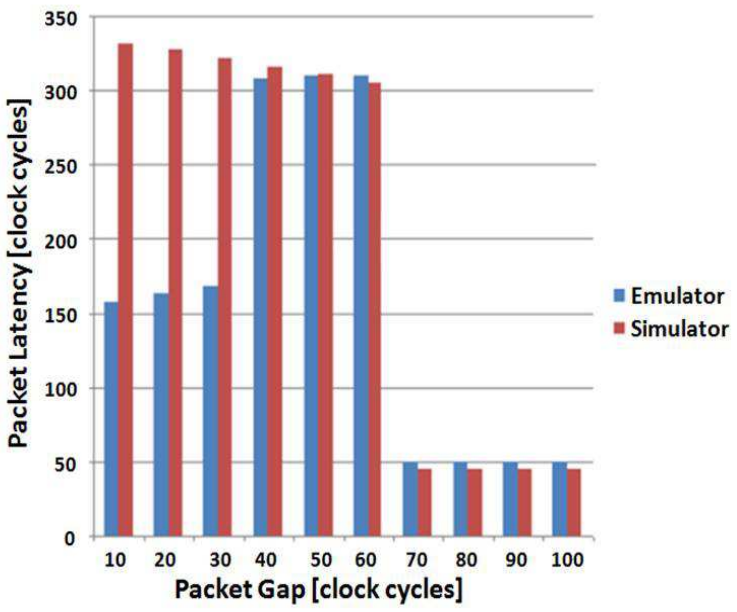


Fig. 46 - Packet latency vs. packet gap.

3.7. Protocol test time comparison

A key issue of a design and verification environment is the time needed to simulate or emulate a given time frame of the real system. Hence, several tests were done to assess the protocol test time of the proposed FPGA emulator compared to the protocol test time achievable with other high performance (HP) computing machines or with a distributed computing GRID. The following protocol test conditions were adopted:

1. Test time: 60s;
2. Speed: 4x, 8x;
3. Trigger rate: 133 kHz, 400 kHz;
4. Packet rate: 133 kHz, 400 kHz;
5. Packet size: 5 or 8 words (16bit).

The HP machines used for the comparison are:

1. Intel Core Duo 2 T9300 (45 nm CMOS technology) processor with a 2.5 GHz clock speed, 2 cores, 6 MB of L2 cache and a 64 bit instruction set;
2. Intel Xeon X7550 (45 nm CMOS technology) processor with a 2.0 GHz clock speed, 8 cores, 18 MB of L3 cache and 64 bit instruction set;
3. INFN (Istituto Nazionale di Fisica Nucleare) PISA CSN4Cluster data center [30], formed by a network of 256 AMD Opteron 2356 QUADCORE 2.3 GHz processors with a total elaboration power of 10 Terafloat/s and a data storage capacity of 10 TeraBytes.

In Table 8 the achieved test timing results are summarized. It is worth noting that by using the FPGA emulator platform very fast tests can be performedconducted: 1 minute of real system time can be emulated in roughly 6 minutes and so very long experiments in great details can be carried out. Thanks to our platform, the test time is decreased by a factor of about 20x and 100x compared to HP computing machines such as the 8-core Intel Xeon and the 2-core Intel CoreDuo respectively. The testing time performance of the proposed emulator platform is comparable to the results achievable with a much complex and costly 256 CPU GRID computing center [30].

Table 8 - Test Time Comparison.

Test Environment	**Test Time**
Simulation on Intel Core Duo@2.5GHz	**11h 47m**
Simulation on Intel Xeon 8-Core@2.0GHz	**2h 21m**
Simulation on INFN CSN4Cluster (256 CPUs)	**3m 21s**
FPGA Emulator	**6m 40s**

3.8. Conclusions and future developments

In this chapter an emulation platform for the verification and validation of FF-LYNX protocol interfaces was introduced. The FF-LYNX protocol is fast and flexible and is utilizable for several applications in HEP, medical and space research fields. The design of this platform is one phase of the top-down design flow of the ASIC chip which implements the FF-LYNX protocol interfaces. The proposed FPGA-based platform allows a deeper and faster verification of FF-LYNX protocol interface model with respect to high level simulations conducted with ISE platform. The verification time was decreased of about fifty times less than high level simulator of ISE platform. In addition, the System-C models, designed during the protocol definition phase, was quite good for validate the protocol functionality but it did not model the hardware protocol interface behavior perfectly, so it was necessary to use a VHDL language for the design, avoiding every model error. After the verification conducted with this platform, the ASIC chip was designed using a IBM 180nm technology. Thanks to the flexibility of this platform, it was possible to modify the emulation system to be used as test bed to test the proper working of the ASIC chip, both in standard conditions and during X-ray irradiations to verify its hardness against the TD (Total Dose). In the future, this platform will be modified to be a test bed for the ASIC chip during heavy ions irradiation in order to test its hardness against SEEs (Single Event Effect).

4. DEVELOPMENT OF A CALCULATION PLATFORM FOR DYNAMICAL SYSTEMS SIMULATION

4.1. Introduction

During the last year of work, a new kind of digital embedded system was designed in order to support investigation and research on complex physical dynamics. Nowadays, in several fields of physics and engineering it is necessary to analyze various complex problems such as numerical differential equation solving, high-definition image processing, target recognition and tracking, etc., in which there are a lot of data to elaborate. So, data handling and elaboration time are important and critical factors.

In order to tackle these problems, higher and higher performance computers, such as multi-core processor (McP) grid could be used. This approach is not very efficient because of the complexity in programming parallel code, the managing and controlling of a large grid and the necessity to have access to such facilities. Besides, these McP grids are expensive and power-hungry and it is impossible to use those in embedded applications such as industrial controllers, missile guidance systems, video surveillance, etc.

It is possible to use several hardware technologies such as digital signal processing (DSP) devices [31], reconfigurable digital devices (e.g.: FPGAs) [32], [33], [34], symmetric multi-processing (SMP) machines [35] in order to solve that problem. Neural network systems [36] and cellular neural network (CNN) paradigms [37], [38] are employed to achieve the purpose.

The CNN architecture, which will be introduced in 4.2 section, takes advantage over the neural network one thanks to its lower implementation complexity. It is convenient to investigate locally interconnected dynamical structures thanks to the CNN distributed computing approach. The evolution of this idea is the cellular neural network universal machine (CNN-UM) which is a new CNN computing structure formed by an array of NxM dedicated processors [39]. The CNN paradigm can be implemented in many kind of technologies such as analog devices [40], [41], hybrid digital devices, embedded architectures, digital signal processing systems and reconfigurable platforms as FPGAs [42]. Clearly, there are pros and cons in the choice of each technology. By using the analog approach there is the advantage to have higher computing performances but a lower accuracy due to non-linearity and dispersion of analog component parameters. With a reconfigurable digital approach we have lower design costs, higher accuracy but lower elaboration performances.

Recently, there is the possibility to use graphic processing unit (GPU) systems, thanks to programming environments such as the Compute Unified Device Architecture (CUDA). This approach led to significant improvements in data managing and elaboration speed due to the highly parallelized hardware architecture typical of graphic processors [43]. Speedup ratios ranging from tens to hundreds, w.r.t. standard CPU approach, were demonstrated in several application fields such as image processing [44] and fluid dynamic simulations [45]. GPU are relatively low cost and natively integrated in common PC platforms but their architecture is, as expected, optimized for the algorithms to be executed by the PC graphic board. So the implementation of the CNN-UM always requires the re

organization of the algorithm in order to efficiently use the GPU resources in terms of number of parallel processes and internal data transfer rate [46].
In this chapter an FPGA based distributed computing microarchitecture (DCMARK) based on the CNN-UM paradigm is proposed. In literature there are many similar approach in several research fields [47], [48], [49], [50] in confirmation of its efficiency. The guiding idea was to push the number of digital computing units, working in parallel, to the limit of the CNN cell's number, replicating the CNN architecture, taking advantage, at the same time, of the efficiency of the hardware implementation of the local interconnection. The choice of an FPGA platform for the development phase was obvious, but the possibility to maintain a high degree of reconfigurability in the system convinced us to select it as the final implementation technology.
A calculation system (DCMARK Calculator) was developed to this aim, of which DCMARK is the computing part. This system is used to solve partial differential equation systems, in particular, as a well known benchmark, we chose the one dimensional Korteweg de Vries equation system [51]. Using this type of architecture it is possible to reduce the elaboration time, increase the CNN array size (number of computing cells), increase equation solution accuracy and obtain a run-time fast calculator.

4.2. Cellular Neural Networks (CNNs) basis

A cellular neural network (CNN) is an artificial neural network which features a multi-dimensional array of neurons and local interconnections among the cells. The original CNN paradigm was first proposed by Chua and Yang in 1988. The two most fundamental ingredients of the CNN paradigm are: the use of analog processing cells with continuous signal values, and local interaction within a finite radius. A CNN is a nonlinear analog circuit which processes signals in real time. It is made of a massive aggregate of regularly spaced cloned circuit, called cells, which communicate with each other directly only through their nearest neighbors.

4.2.1. Architecture of CNNs

Any cell in a CNN is connected only to its neighbor cells. The adjacent cells can interact directly with each other. Cells not directly connected together may affect each other indirectly because of the propagation effects of the dynamics of CNNs. An example of a two-dimensional CNN is shown in Fig. 47.

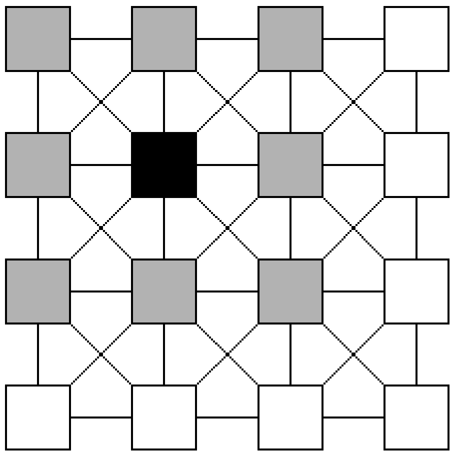


Fig. 47 - Cellular Neural Network schematic structure.

Every cell is influenced by a limited number of cells in its environment. This locality of connections between the units is the main difference between CNNs and other neural networks. Large CNN chips can be implemented using VLSI techniques. The Fig. 44 shows the emphasized cell (black) connected to the nearest neighbors (gray). The cells marked in gray represent the neighborhood cells of the black cell. The neighborhood includes the black cell itself. This is called a "3*3-neighborhood". Similarly, we could define a "5*5-neighborhood", a "7*7-neigborhood" and so on.
The basic circuit unit of CNNs is called a cell. It contains linear and nonlinear circuit elements, which typically are linear capacitors, linear resistors, linear and nonlinear controlled sources, and independent sources. All the cells of a CNN have the same circuit structure and element values. A typical circuit of a single cell is shown in Fig. 48.

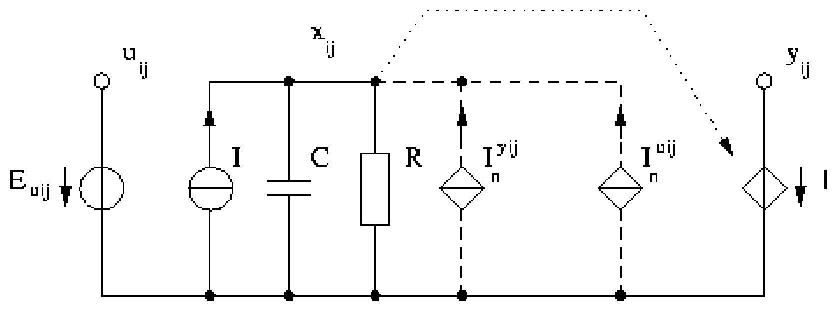


Fig. 48 - Typical circuit of a single cell.

Each cell contains one independent voltage source $E_{u\,ij}$ (Input), one independent current source I (Bias), several voltage controlled current sources $I_n^{u\,ij}$, $I_n^{y\,ij}$, and one

voltage controlled voltage source $E_{y\ ij}$ (Output). The controlled current sources $I_n^{u\ ij}$ are coupled to neighbor cells via the control input voltage of each neighbor cell. Similarly, the controlled current sources $I_n^{y\ ij}$ are coupled to their neighbor cells via the feedback from the output voltage of each neighbor cell.
The cell C(i,j) has direct connections to its neighbors through two kinds of weights: the feedback weights a(k,l;i,j) and the control weights b(k,l;i,j), where the index pair (k,l;i,j) represents the direction of signal from C(i,j) to C(k,l). The coefficients a(k,l;i,j) are arranged in the feedback-Template or A-Template. The coefficients b(k,l;i,j) are arranged in the control-Template or B-Template. The A-Template and the B-Template are assumed to be the same for all the cells in the network. The global behavior of a CNN is characterized by a Template Set containing the A-Template, the B-Template, and the Bias I. If we assume a "3*3-neighborhood", the Template Set consists of 19 coefficients. The external input to the cell is typically assumed to be constant over a certain operation interval. Therefore, the total input current to the cell is given by the weighted sum of control inputs and weighted sum of feedback outputs. In addition, a constant bias term (I) is added to the cell. Due to the capacitance C and resistance R, the state voltage x(i,j) satisfies the following differential equation:

$$C\frac{dx_{i,j}}{dt} = -\frac{1}{R}x_{i,j} + \sum_k (a_k y_k + b_k u) + 1 \qquad (7)$$

k denotes the neighborhood of the specific cell

Without loss of generality, the time constant T = R*C can be set to 1.
The only nonlinear element in each cell is a piecewise-linear voltage controlled voltage source with characteristic

$$y(i,j) = f(x(i,j)) \qquad (8)$$

A widely used nonlinearity is the piecewise-linear function (Fig. 49) as given by:

$$y(i,j) = f\big(x(i,j)\big) = 0.5 * (|x+1| - |x-1|) \qquad (9)$$

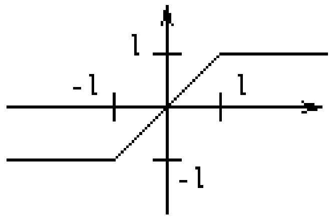


Fig. 49 - The piecewice-linear function.

The block diagram of a cell C(i,j) is shown in the Fig. 50.

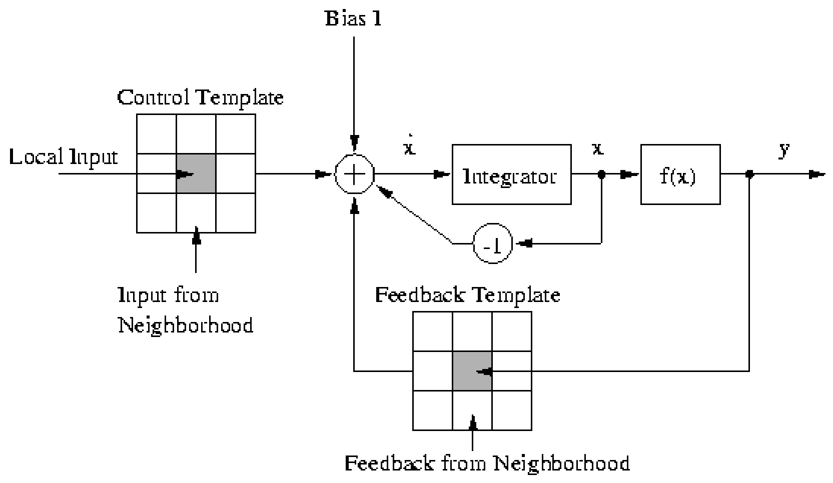


Fig. 50 - A block diagram of a cell C(i,j).

4.2.2. Global behavior of CNNs

In image processing, n-by-m rectangular grid arrays are often used. n and m are the numbers of rows and columns, respectively. Each cell in a CNN corresponds to an element of the array. Assuming that each cell is connected to its nearest neighbors only ("3*3-neighborhood") and that the local connections of a cell do not depend on the cell's position, the Template set contains 19 coefficients (A-Template: a1 .. a9, B-Template: b1 .. b9, Bias I). The behavior of the CNN is completely determined by this Template set.
New CNN-Templates for arbitrary tasks may be found using a training algorithm, or by defining local rules for a given global task. The local rules describe a cell's equilibrium state depending on the inputs and outputs of the neighbor cells. The inputs and the outputs of the neighbor cells are assumed the be constant. The dynamics of the cell is not specified. If template values for the local rules are found, simulations are very helpful to test the dynamic global behavior of the entire clone of cells.
Optimal coefficient calculation leads to solutions which converge after short time. This means that the output of every cell reaches its final output y=+1 or y=-1 after short time.

4.2.3. Possible applications

CNNs can be used in many scientific applications:
In signal processing, CNNs show great promise in solving many complex problems that cannot be solved satisfactorily using conventional approaches.

1. solve the maximum-likelihood estimation of signals in the presence of intersymbol interference and white Gaussian noise

In image processing that deals with gray-scale image inputs, CNNs can be applied to perform

1. Feature extraction & classification;
2. Motion detection & estimation;
3. Collision avoidance;
4. Object counting and size estimation;
5. Path tracking.

In analyzing 3-D complex surfaces, the CNN is capable of

1. Detecting minima and maxima;
2. Detecting area with gradients that exceed a given threshold.

In solving partial differential equations, CNN is suitable for reducing non-visual problems to geometric maps for

1. Thermographic maps;
2. Antenna-array images;
3. Medical maps and images.

4.3. General system architecture

The complete block diagram of the whole calculation system is shown in Fig. 51. It is divided into three main parts:

1. a Terasic DE4-230 FPGA Development Board;
2. the System on Programmable Chip (SoPC) which contains our Distributed Computing Microarchitecture block implemented on the FPGA;
3. an host PC with a Graphic User Interface (GUI) for calculation management.

The Development board and the host PC communicate using an Ethernet protocol connection.

4.3.1. DE4-230 FPGA development board

The DE4-230, from Terasic Tech.inc, is an FPGA Development Board (Fig. 52) equipped with an Altera Stratix IV GX EP4SGX230 FPGA featuring about 228,000 Logic Elements (LEs), 91,200 Adaptive Logic Modules (ALMs), 14,283kb Embedded Memory and 8 Phase Locked Loops (PLLs). The board includes: 64MB Flash memory, 2MB SSRAM memory, 2kb EEPROM memory, two DDR2 memory slots up to 8GB, three USB 2.0 ports, RS-232 port, four Serial ATA 3.0 ports, four Gigabit Ethernet ports, a PCI Express x8 edge connectors (up to 5.0Gbps/lane for Gen2), SD card slot, 9 SMA connectors. As it can be shown in the following, the board demonstrated to be a powerful platform for the implementation and test of the DCMARK idea and is ready for future evolutions.

4.3.2. System on programmable chip (SoPC)

SoPC is a complete system controlled by an Altera NIOS II IP softcore processor. In the system there are: a Triple Speed Ethernet (TSE) module for communications, our Distributed Computing System (DCSYS), and Scatter-Gather (SG) DMA modules for fast transmission operations between devices. NIOS II processor manages all main SoPC operations such as communications between

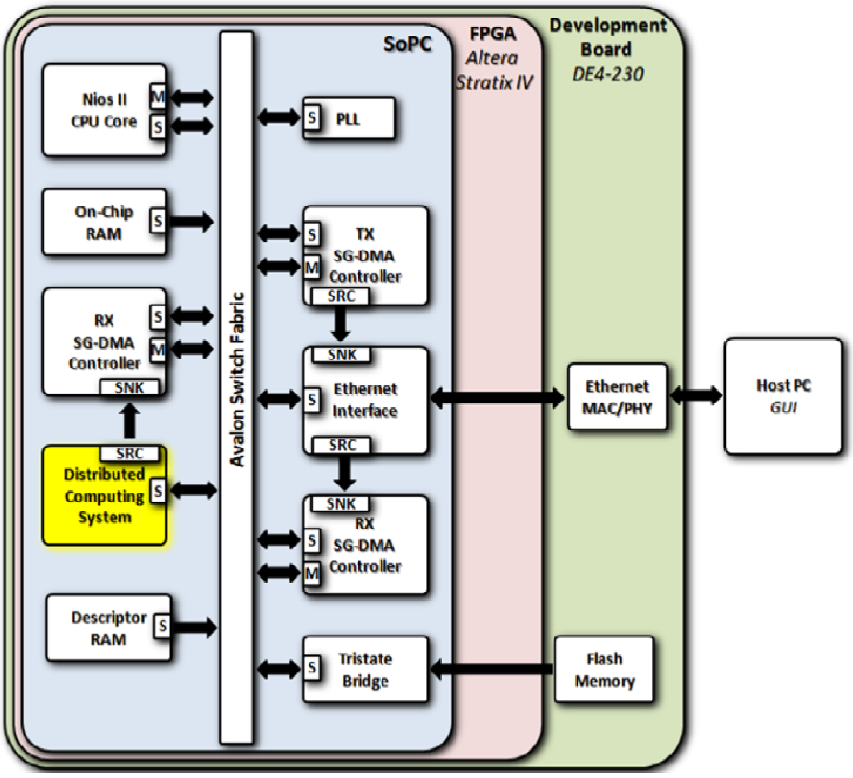


Fig. 51 - DCMARK Calculator block diagram.

TSE module and the host PC, devices interrupt handling, MicroC/OS II Operative System (OS) supervising, etc. NIOS II is programmed by the user using the Eclipse software, based on C language. The SoPC allows to interconnect system devices thanks to an Altera communication bus facility called Avalon. There are several kinds of Avalon buses for every need such as Memory-Mapped (MM) bus with a Master (M) and Slave (S) structure for device command and control operations and STreaming (ST) bus with a Source (SRC) and Sink (SNK) structure for continuous data transmissions.

Fig. 52 - DCMARK Calculator block diagram.

4.4. *Distributed computing microarchitecture (DCMARK)*

The DCMARK, included into the Distributed Computing System (DCSYS) block of SoPC, is based on the CNN-UM approach in which several custom processors execute a group of sequential operations at the same time in order to elaborate particular information. At the first computing step each processor acquires status data from the neighboring processors (with a limited sphere of influence) and after a determined number of clock cycles (elaboration time) they will give their local results. These results can be analyzed versus time and/or space. With a time analysis we study just one single processor (a fixed spatial point) results versus time while with a space analysis we study all processors (all spatial points) results at a fixed time step.

In this paper, by way of example, a 1-D locally interconnected dynamical system is investigated. This system is based on a discretized partial differential equation where every point of the spatial array is a dynamical element. Each processor of DCMARK is dedicated to a dynamical element of the spatial array. In our case every dynamical element has a neighborhood formed by four dynamical elements (two on its right and two on its left).

Table 9 - DCMARK microinstructions

Instruction	OpCode	N° Clock Cycles	Function
FETCH	000000	2	Fetching operation
LDA	000001	2	Loading data in Reg A
LDB	000010	2	Loading data in Reg B
LDI	000011	2	Loading Status Cell in Reg I
ST	000100	1	Storing Result on a Data RAM location
STM2	000101	2	Storing M2 Reg on M2 RAM location
STM1	000110	2	Storing M1 Reg on M1 RAM location
STP1	000111	2	Storing P1 Reg on P1 RAM location
STP2	001000	2	Storing P2 Reg on P2 RAM location
ADD	001001	8	Adding Reg A to Reg B
SUB	001010	8	Subtracting Reg B from Reg A
MUL	001011	6	Multiplying Reg A by Reg B
JUMP	001100	1	Jumping to a micro-code RAM Location

4.4.1. Single cell block

Each calculation unit of DCMARK is called a Cell and is based on a Von Neumann elaboration architecture, thus its RAM memory stores both Data and Micro-Code (MCode). The MCode approach was chosen in order to easily modify the physical phenomenon investigation just changing the implemented equation (as long as it is expressible with sums and multiplications).
The Cell (Fig. 54) has a 40bit Data Bus and an 8bit Address Bus. Its Arithmetic Logic Unit (ALU) allows to execute Floating Point (FP) additions and multiplications. In order to demonstrate the DCMARK idea, this first implementation of the Cell uses ALTERA blocks for the adder and multiplier, leading to a maximum amount of about 200 Cells to be integrated in the FPGA device adopted. This number is expected to be substantially increased by working on the customization of the ALU block and thanks to the continuing growth of the available device size.
The Cell, that can be clocked up to 180MHz, contains:

1. a Control Module, implemented as a Finite State Machine (FSM) which controls the micro-code execution and enables the control signals;
2. a 40bit x 256 RAM memory;

3. a 32bit Floating Point (FP) Adder;
4. a 32bit Floating Point (FP) Multiplier;
5. a 8bit Program Counter;
6. a 40bit Instruction Register;
7. three 32bit Operation Registers (A, B and C) for arithmetical operations;
8. five 32bit input/output (I/O) Registers (I, M2, M1, P1, P2) for acquiring current status data from its neighborhood;
9. a 6x1 40bit Data Multiplexer;
10. a 2x1 8bit Multiplexer;
11. a 2x1 32bit Result Multiplexer.

The Micro-Code is written using a group of 13 custom micro-instructions as in Table 9.
Each instruction stored in RAM has the following format (Fig. 53):

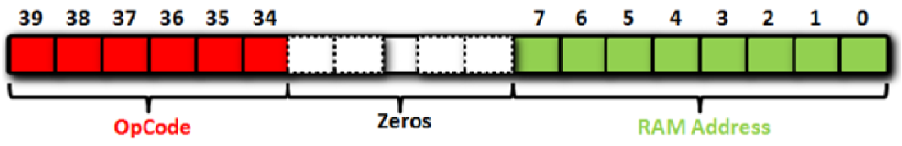


Fig. 53 - RAM data word structure.

4.4.2. Parallel cell configuration module

In order to quickly program the RAMs of all the Cells, we designed a Parallel Cell Configuration Module (PCCM) (Fig. 55).
The PCCM is formed by:

1. a Configuration Block: it is a Finite State Machine (FSM) which reads a configuration File from Configuration ROM and programs all the Cells;
2. a Configuration ROM: it stores a Configuration File containing MCode, initial status variables and constants;
3. a Write Decoder: it allows to address every Cell for one-to-one and parallel programming. One-to-one programming allows to store a different initial status variable on each cell RAM while parallel programming allows to store MCode and constants on all cell RAM at the same time.

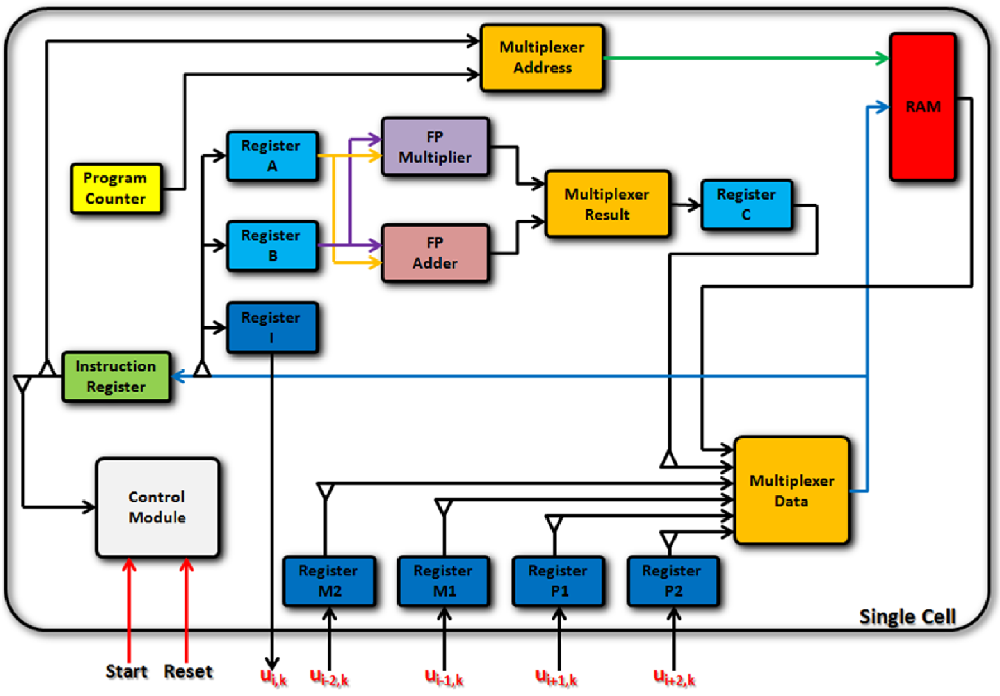


Fig. 54 - DCMARK single cell block diagram (working registers are in sky-blue and I/O registers are in dark blue; the blue and green lines represent the 40bit data bus and the 8bit address bus respectively).

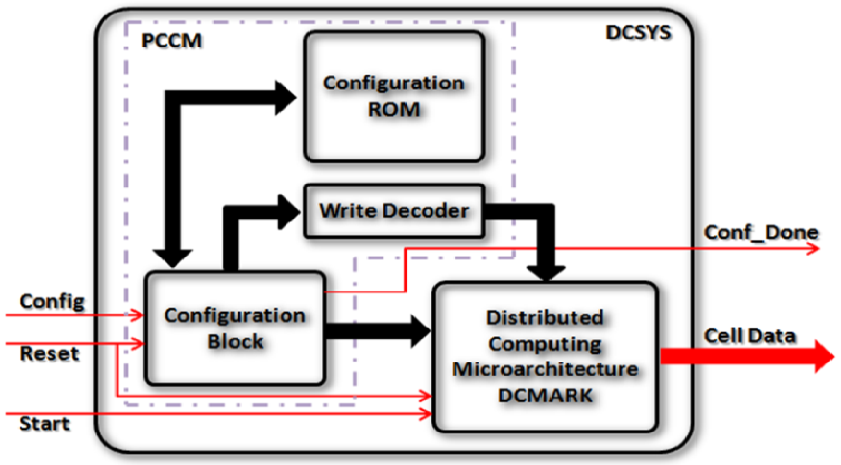


Fig. 55 - DCSYS block diagram.

4.5. *Complex physical dynamics investigation*

One of the most important topic of contemporary science focuses on the study of continuous [52],[53] and discrete [54], [55] dynamical systems, analyzing their organization as non linear evolving structures [56]. Chaos is the most striking feature of their behavior. The concept of dynamical system is connected to a mathematical model which describes its time evolution and it is often characterized by differential equations [57]. Differential equation solving allows to define and forecast the future evolution of system in time and space. To allow a more and more detailed analysis of dynamic systems it is absolutely necessary to perform long and heavy numerical simulations which would require powerful, fast and expensive elaborators (sometime multi-core grid). In order to verify the quality of our DCMARK approach we began to investigate a no complex problem characterized by a one-dimension partial differential equation.
A case study: 1-D Korteweg de Vries equation
The Korteweg de Vries takes its name from Diederik Korteweg and Gustav de Vries who, in 1895, proposed a mathematical model which allowed to predict the waves behaviour on shallow water surfaces [51]. The solutions of this equation were self-reinforcing solitary waves named Solitons and had several interesting properties. Mainly, these solutions are permanent shape, and localized within a region and when they interact with other solitons they don't change their speed or shape (neither a signal amplification or signal fading) but they just have a phase shift [58], [59]. There are many research topics explained by the KdV equation, such as the already mentioned shallow-water waves [60], the ion-acoustic waves in plasma [61], [62], the wave propagation in nonlinear lattice [63], the non-linear transmission networks [64], [65] and the Fermi-Pasta-Ulam recurrence problem [66].
The main idea is modulating solitons and transmitting them on communication lines such as optical fibers.
The one-dimension (1-D) Korteweg de Vries differential equation is the following [63]:

$$\frac{\partial u(t,x)}{\partial t} = -6u(t,x)\frac{\partial u(t,x)}{\partial x} - \frac{\partial^3 u(t,x)}{\partial x^3} \qquad (10)$$

Where u(t,x) is the solitonic propagating wave. The progressive wave (called soliton) of the KdV equation has the following expression:

$$u(t,x) = -\frac{\nu}{2}\cdot \operatorname{sech}^2\left[\frac{\sqrt{\nu}}{2}(x - \nu t - x_0)\right] \qquad (11)$$

Where ν is the wave velocity and x_0 is the initial spatial constant.
Furthermore, the KdV equation can be analytically solved by the inverse scattering transform [67], [68]

4.5.1. Discretization of KdV equation

In order to implement the equation on DCMARK we had to discretize (10). Considering the second term in right-end side of (10) we can lay down:

$$u(t,x)\frac{\partial u(t,x)}{\partial t} = \frac{1}{2}\frac{\partial [u(t,x)]^2}{\partial x} \qquad (12)$$

hence, as in [62], the (10) becomes:

$$\frac{\partial u(t,x)}{\partial t} = -3\frac{\partial [u(t,x)]^2}{\partial x} - \frac{\partial^3 u(t,x)}{\partial x^3} \qquad (13)$$

We used for the numerical discretization of spatial derivative terms of (13), a space-centered finite difference method [69] and we divided the KdV equation in N single equations [40]:

$$\frac{\partial u_i}{\partial t} = \frac{1}{2\Delta x^3}[(u_{i-2} - u_{i+2}) + 2(u_{i+1} - u_{i-1})] + \frac{3}{2\Delta x}({u_{i-1}}^2 - {u_{i+1}}^2) \qquad (14)$$

where i=0,...,N. are the space iteration index and Δx is the space step of the discrete grid.

For the time derivative term of (14), just for the first iteration, we used a forward time finite difference method (15) as in [59], [61] because there is no preceding value at the first step of numerical integration process. Hence, for the other iterations, we used a centered-time finite difference method (16). We set $K_{i1} = \frac{1}{2\Delta x^3}$, $K_{i2} = \frac{3}{2\Delta x}$ and $K_1 = \frac{1}{\Delta x^3}$, $K_2 = \frac{3}{\Delta x}$

$$u_i^{k+1} = u_i^k + \Delta t\left\{K_{i1}\left[\left(u_{i-2}^k - u_{i+2}^k\right) + 2\left(u_{i+1}^k - u_{i-1}^k\right)\right] + K_{i2}\left({u_{i-1}^k}^2 - {u_{i+1}^k}^2\right)\right\} \qquad (15)$$

$$u_i^{k+1} = u_i^{k-1} + \Delta t\left\{K_1\left[\left(u_{i-2}^k - u_{i+2}^k\right) + 2\left(u_{i+1}^k - u_{i-1}^k\right)\right] + K_2\left[{u_{i-1}^k}^2 - {u_{i+1}^k}^2 + u_i^k\left(u_{i-1}^k - u_{i+1}^k\right)\right]\right\} \qquad (16)$$

where k=0,...,M. are the time iteration index, i=0,...,N. are the space iteration index and Δt is the integration time. Using this combined approach we have a stable loop propagation of a soliton through all cells for all time cycles.

This kind of discretization is less accurate than other types but it is also the best technique in terms of implementation easiness and resources saving on embedded systems.

The linchpin of the calculator idea is to consider every single u_i with i=0,...,N. a single solitonic state cell which calculates its future state value on the basis of state values of its first and second neighbors that is $u_{i\mp a}$ with a=1,2 as in [41], [71].

4.6. KdV implementation on DCMARK

The implementation of KdV equation on DCMARK consists of two main implementation steps: a MCode step and a Cells Network step.

4.6.1. MCode implementation step

MCode step consists of dividing (15) and (16) in single micro-instructions to be stored on RAM.
We defined 14 arithmetical operations for (15), where ROpx is the operation result which is stored on RAM:

1. Opi1 : $(u_{i-2} - u_{i+2})$ => ROp1
2. Opi2 : $(u_{i+1} - u_{i-1})$ => ROp2
3. Opi3 : $(\mathrm{ROp2} + \mathrm{ROp2})$ => ROp3
4. Opi4 : $(\mathrm{ROp1} + \mathrm{ROp3})$ => ROp4
5. Opi5 : $(\mathrm{K}_1 * \mathrm{ROp4})$ => ROp5
6. Opi6 : $(u_{i-1} * u_{i-1})$ => ROp6
7. Opi7 : $(u_{i+1} * u_{i+1})$ => ROp7
8. Opi8 : $(\mathrm{ROp6} - \mathrm{ROp7})$ => ROp8
9. Opi9 : $(\mathrm{K}_2 * \mathrm{ROp8})$ => ROp9
10. Opi10 : $(\mathrm{ROp5} + \mathrm{ROp9})$ => ROp10
11. Opi11 : $(\Delta t * \mathrm{ROp10})$ => ROp11
12. Opi12 : $(u_i + \mathrm{ROp11})$ => ROp12
13. Opi13 : $(u_i + \mathrm{ZERO})$ => (updating u_i^{k-1})
14. Opi14 : $(\mathrm{ROp12} + \mathrm{ZERO})$ => (updating u_i)

For (16) we defined 17 arithmetical operations, but the first eight are the same as those for (15):

1. Op9 : $(u_{i-1} - u_{i+1})$ => ROp9
2. Op10 : $(u_i * \mathrm{ROp9})$ => ROp10
3. Op11 : $(\mathrm{ROp8} + \mathrm{ROp10})$ => ROp11
4. Op12 : $(\mathrm{K}_2 * \mathrm{ROp11})$ => ROp12
5. Op13 : $(\mathrm{ROp5} + \mathrm{ROp12})$ => ROp13
6. Op14 : $(\Delta t * \mathrm{ROp13})$ => ROp14
7. Op15 : $\left(u_i^{k-1} + \mathrm{ROp14}\right)$ => ROp15
8. Op16 : $(u_i + \mathrm{ZERO})$ => (updating u_i^{k-1})
9. Op17 : $(\mathrm{ROp15} + \mathrm{ZERO})$ => (updating u_i)

The Opi13, Opi14 and Op16, Op17 have the task to update the Cell Status variables at the end of every iteration, that is the old value of u_i becomes u_i^{k-1} and the new value of u_i is updated.
After the arithmetic operations definition we started to write all MCode copies according to a well-defined process. This process starts loading the Cell Status variable u_i^k, from RAM and storing it on I/O Register I to be available for other neighbor Cells and then storing on RAM the four neighbor Cell Status variables $u_{i-2}^k, u_{i-1}^k, u_{i+1}^k, u_{i+2}^k$ stored in I/O Registers M2, M1, P1 and P2, respectively. After every iteration step this process is re-executed.

This loading/storing operations are conducted using the micro-instructions in Table I, that is LDI, STM2, STM1, STP1 and STP2.
In the RAM structure, shown in Fig. 56, we find four main parts: Micro_Code part to store the 137 micro-instructions, Status Variables part to store the Cell Status variables, Constants part to store the constants defined in (15) and (16) and Operation Variables to store the partial operation results. We have also four free locations for possible modifications.

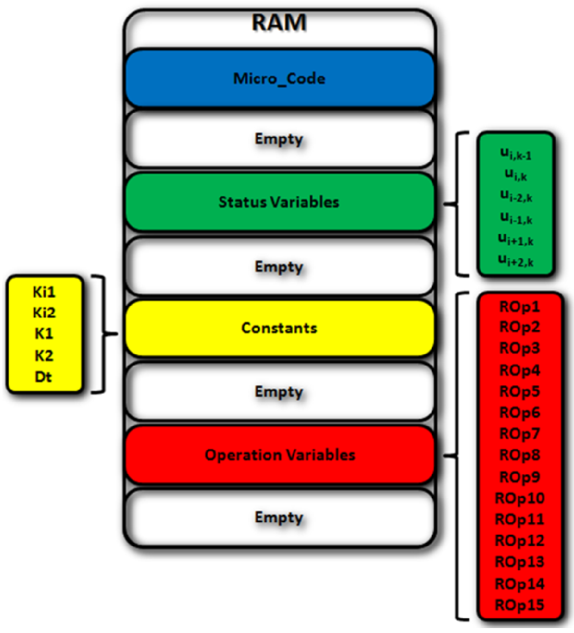


Fig. 56 - RAM structure.

The MCode is formed by 137 micro-instructions but after the system start-up (after the first iteration) only 74 micro-instructions are executed in the computing loop.
The content of Configuration File stored on Configuration ROM has the structure as shown in Fig. 57.

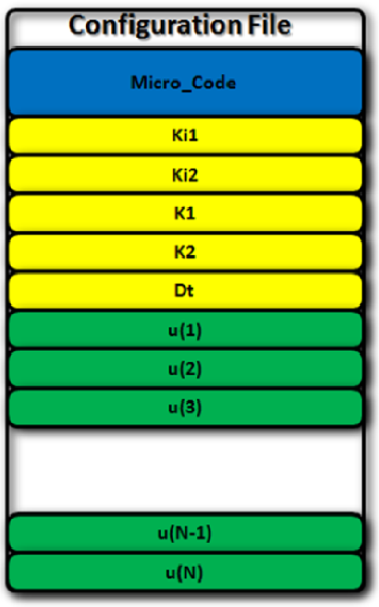


Fig. 57 - ROM Configuration File structure.

4.6.2. Cells network implementation step

Cells Network step lies in connecting properly every Cell with its first and second neighbors according to Cell relationship shown in (15) and (16).
In particular, using the approach in [41], we connected the Cells building a Ring network as in Fig. 58.

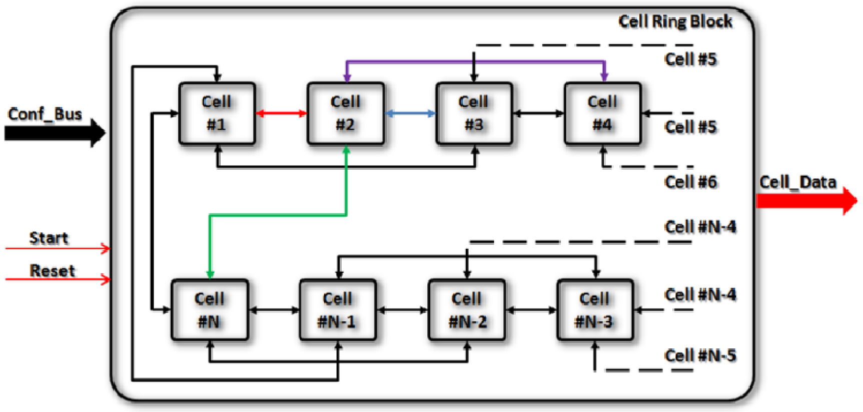


Fig. 58 - Cell Ring Block diagram (for Cell #2 case: red line shows the link to first left neighbor, blue line the link to first right neighbor, green line the link to second left neighbor and violet line the link to second right neighbor).

4.6.3. DCMARK performances and used resources

The preliminary version of a Single Cell processor and of DCMARK is implemented on FPGA Stratix IV GX. In Table 10 we find the resources used, without any kind of design optimization.
As regards the Single Cell computing time, every Cell executes a KdV iteration (integration step) producing a 32bit output value in about 3,77μs with a 100MHz system clock hence with a throughput of about 8,488,063 bit/s. But considering a Cell Ring of N Single Cells this throughput has to be multiplied by N.
Using the FPGA Altera Stratix IV GX, we can implement up to 200 Single Cells on our DCMARK.

Table 10 - Single cell used resources

FPGA Resources	**Used Resources**
ALMs	561
Combinatorial ALUTs	860
Total Registers	800
Total Block Memory bits	10,240
DSP Block 18-bit elements	4
DSP 36x36	1

4.7. Analysis settings and results

We executed two kinds of analysis:

1. a high-level test by means of a MatLab software simulator in order to verify the quality of equation discretization and to study the variation of parameters such as number of Cells, Δx, Δt, Initial Cell Status and number of iterations.
2. a calculation test using the whole calculation system to verify the correctness of results compared to simulation results and elaboration time.

4.7.1. KdV simulation test

The first parameters to tune are Δx and Δt. According to [61] these two parameters have to be related on the basis of (17), called Courant-Friedrichs-Lewy (CFL) condition, to have convergence.

$$v \cdot \Delta t / \Delta x \leq C \qquad (17)$$

where v is the wave velocity by which the wave goes from x_i to x_{i+1} and C is a constant which depends on the equation. In a nutshell, Δt has to be smaller than $\Delta x / \nu$.
Then we chose the number of Cells closed in the Ring Network and the Initial Cell Status.
A hyperbolic secant squared function is chosen:

$$\mathrm{u_i} = \mathrm{K} \cdot \mathrm{sech}^2\, x_i \text{ with } min < x_i < max,$$
$$0 < i < (max - min)/\Delta x,\ \Delta x = x_{i+1} - x_i\ .$$

This function avoids divergence integration problems, thanks to its zero-tangent envelope for $x \to \pm\infty$.
We conducted three types of simulations: time, space and time/space simulations.
In the following figures we can see some simulation results with the parameter settings of Table 11:

Table 11 - Simulation parameters configuration.

Simulation Parameters	**Parameter Values**	**Notes**
N° Cells	100	Ring-like network
Δx	0.5 mm	according to CFL condition
Δt	0.01 s	
N° of Iterations	10000	
Initial Cell Status	$\mathrm{u_i} = 2 \cdot \mathrm{sech}^2 x_i$	$-5 < x_i < 5$

As we can see in Fig. 59 a soliton, starting from the tenth cell, travels through the 100 cell ring network and after about 1500 iterations (15 s) it completes one loop. A soliton travels from a cell to another in about 15 iterations (0.15s) with a steady wave velocity of about 3.3 mm/s.

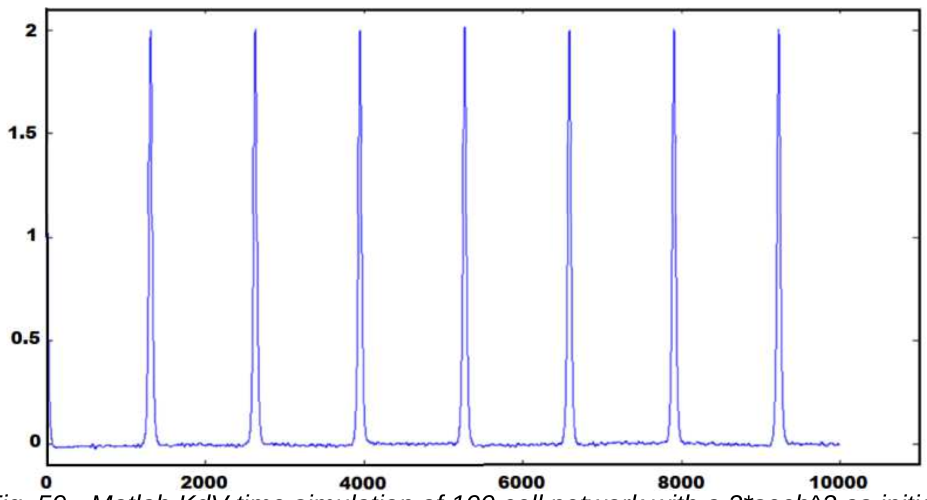


*Fig. 59 - Matlab KdV time simulation of 100 cell network with a 2*sech^2 as initial status function (10th cell output).*

In Fig. 60 there is a time/space graph in which it is clear that soliton travels with a steady wave velocity (constant slope lines) through the cell ring network.

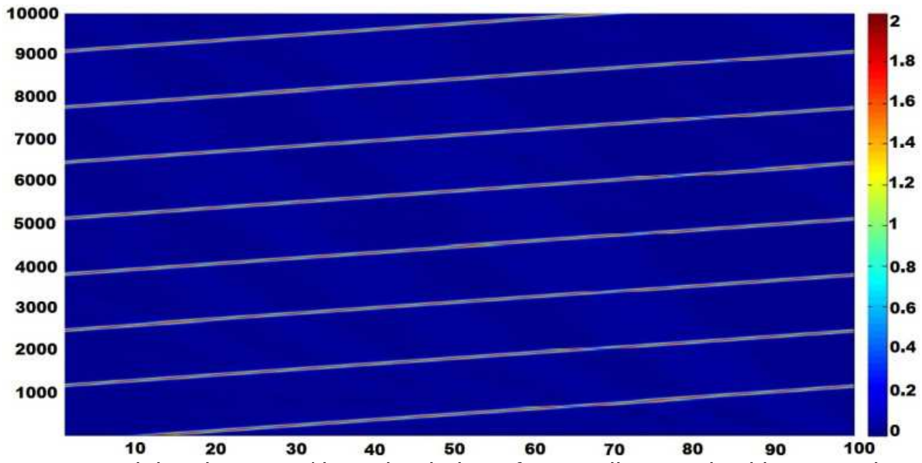


*Fig. 60 - . Matlab KdV space/time simulation of 100 cell network with a 2*sech^2 as initial status function.*

These results confirm both the stability of KdV equation numerical solving after many integration steps and the physical phenomenon emergence of soliton propagation.

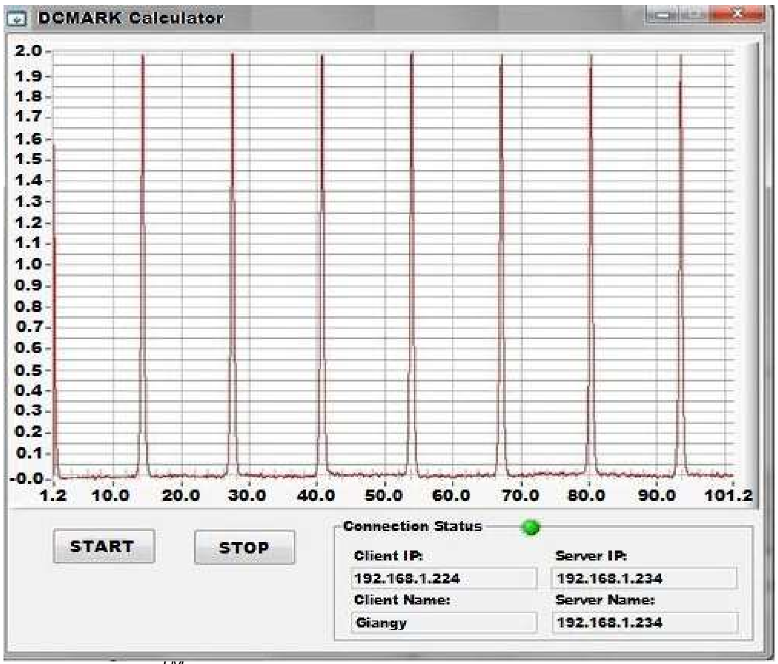
4.7.2. Calculation results

In our tests we deployed up to the maximum number of Single Cells implementable on DCMARK, that is 200 Single Cells. The Single Cell is still a prototype core and so it is not optimized for saving FPGA resources. Many Altera library IP cores (such as floating point adders and multipliers, RAMs, counters, etc) with several unused features are instantiated on Single Cell. Our idea for future developments is to design our own cores in order to significantly decrease the Single Cell FPGA requirements.

Table 12 - Calculation parameters configuration

Calculation Parameters	**Parameter Values**	**Notes**
N° Cells	100	Ring-like network
$K_{i1} = 1/(2\Delta x^3)$	4	$\Delta x = 0.5\ mm$
$K_{i2} = 3/(2\Delta x)$	3	
$K_1 = 1/\Delta x^3$	8	
$K_2 = 3/\Delta x$	6	
Δt	0.01 s	according to CFL condition
Initial Cell Status	$u_i = K \cdot \mathrm{sech}^2 x_i$	K = 2, $-5 < x_i < 5$

As previously said, using the Analysis GUI we monitored the analysis evolution.
In the Configuration File, stored on the Configuration ROM, we set as Initial Cell Status the hyperbolic secant squared function as in simulation tests.
In table IV we show the calculation parameter values for a typical KdV analysis.
Fig.61 shows the LabWindows GUI image plotting a KdV calculation result obtained using the DCMARK Calculator according to the parameters settings in Table 12. Therefore, we noticed that the same result of the Matlab simulation is obtained. This result is also confirmed by a numerical comparison between Matlab and DCMARK data.



Client IP: 192.168.1.224
Server IP: 192.168.1.234
Client Name: Giangy
Server Name: 192.168.1.234

*Fig. 61 - . LabWindows^{TM} GUI with results of a KdV time calculation using a 100 cells DCMARK (10th cell output) with a 2*sech^2 as initial status function.*

4.8. Performance comparison

After the Test Phase (Simulation/Calculation) we underlined the differences between MatLab KdV simulations on PC and KdV calculation on FPGA using our DCMARK approach. As elements of comparison we chose two PCs with the following processors:

1. Intel Core-i7 2630QM, 2GHz clock speed, 4 Cores, 8 Threads, 64bit Instruction Set, 6MB Intel Smart Cache.
2. Intel Pentium M 760, 2GHz clock speed, 1 Core, 1 Thread, 32bit Instruction Set, 2MB L2 Cache.

The parameters settings (Δx, Δt and Initial Cell Status) are those in Tables 11, 12 and as we can see in Table 13 the elaboration time for DCMARK system is about 10 times shorter than Intel Core i7 PC and about 70 times shorter than Intel Pentium M PC already for the 100 cells problem. Doubling the cell number, as expected, the performance gap increases. DCMARK performances are unrelated from the number of cells and so, as long as the FPGA resources are saturated, from the complexity of the investigated problem.

Table 13 - DCMARK calculator performance comparison

Elaboration Device	N° Cells	N° Iterations	Elaboration Time
Intel Core i7	100	300000	13 s
		400000	17 s
		500000	21 s
Intel Pentium M	100	300000	1 m 18 s
		400000	1 m 45 s
		500000	2 m 14 s
FPGA DCMARK Calculator	100	300000	**1.2 s**
		400000	**1.6 s**
		500000	**2 s**
Intel Core i7	200	300000	26 s
		400000	34 s
		500000	42 s
Intel Pentium M	200	300000	2 m 41 s
		400000	3 m 33 s
		500000	4 m 31 s
FPGA DCMARK Calculator	200	300000	**1.2 s**
		400000	**1.6 s**
		500000	**2 s**

In Fig.62, we find linear fitted curves about testing time variation with respect to number of iterations for the two study cases: 100 and 200 cells system. It is again underlined the independence of DCMARK performances from the number of cells.

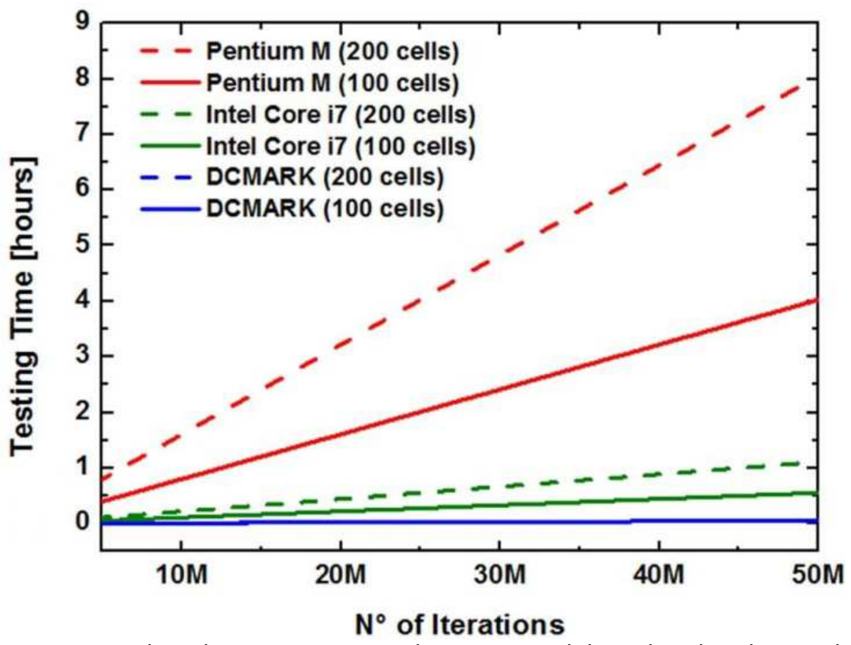


Fig. 62 - . Comparison between PCs and DCMARK elaboration time increasing the number of iterations and the number of Cells .

4.9. Conclusions and future developments

In this chapter we introduced an innovative kind of distributed computing architecture, called DCMARK, for investigating complex physical dynamical problems. DCMARK is the union of a FPGA-based extremely parallelized computing platform and a PC based user interface for setting and analyzing the results of calculations. The main features of this system are the total system reconfigurability for analysing different types of cell-based phenomena and an elaboration time independent from the complexity (in terms of number of cells) of the studied problem.
This hardware calculation approach allows to exploit many concurrent processes executed at the same time, decreasing the elaboration time. Besides, using an FPGA device we exploited its intrinsic reconfigurability and flexibility. The results are promising since, for example, a 100 Cells DCMARK allows to execute KdV equation integration steps 10 times faster than a 4-core processor.
The future development steps to increase the performances will be: the optimization of Single Cell in terms of used resources in order to tackle more and more difficult problems and the improving of the GUI usability. Another important development would be to switch to an ASIC approach, that will lead to loosing reconfigurability but gaining speed, as well as reducing area and improving general system potentiality. Vice versa, taking advantage of the reconfigurability, the DCMARK Calculator can be used in order to implement innovative learning techniques, as in [72] or in analogy to [73].

CONCLUSIONS

This work covered several aspects of design of embedded digital systems for data handling and elaboration. Attention was focused on the design of systems to support both industrial and scientific applications.
At first, an overview of modern embedded systems and a description of the concept of data handling and elaboration were presented. Then the three different embedded platforms were introduced and described in detail.
In the first chapter, the microcontroller-based inertial/GPS platform was introduced. This system allows a smart management and creaming off of digital sensor data, thanks to a proper cooperation of two microcontrollers, and improves the wireless link throughput by means of an efficient buffering and transmission of selected data to host PC. The test demonstrated also the good performance of system with regard to trajectory tracking, using a sensor data-fusion approach, and the battery autonomy. The possibility to integrate all components of this platform on a single dual-layer PCB board would allow to use the system for several different applications such as body motion recognition, fall detection, etc.
In the second chapter, the FPGA-based emulation platform was described. The system was thought firstly to conduct a validation of FF-LYNX protocol interfaces, deeper and faster than the System-C simulations executed on high performance PC. The platform allowed to verify the FF-LYNX interface working up to fifty times more quickly than high level simulator. Thanks to this validation, VHDL FF-LYNX interface models were implemented on ASIC chip for the last phase of design flow. The flexibility of the platform guaranteed, with the right modifications, to use that as test bed for verify the proper working of ASIC chip.
In the last chapter, another FPGA-based platform was treated. This calculation platform was designed for executing complex calculations, in particular to solve non-linear differential equations. The innovative approach was based on a distributed computing micro-architecture which follows a CNN structure. Using this architecture, it is possible to decrease the calculation time up to ten times less than modern multi-core processors. The main result was that the calculation time was almost unrelated from the size and complexity of spatial and temporal grid. The possibility to use larger and larger FPGAs, would guarantee to tackle more and more complex problems and so to improve the computing power of platform.

REFERENCES

1. H. B. Mitchell, **Multi-sensor data fusion – an introduction***,* Springer-Verlag, 2007.
2. S. Heath, **Embedded systems design**, Newnes, 2003.
3. G. J. Lipovski, **Introduction to microcontrollers**, Elsevier, 2004.
4. H. F. W. Sadrozinski, J. Wu, **Applications of field-programmable gate arrays in scientific research**, Taylor & Francis, 2010.
5. J. G. Proakis, D. G. Manolakis, **Digital signal processing: principles**, **algorithms and applications**, Pearson, 2006.
6. A. A. Jerraya, W. Wolf, **Multiprocessor systems-on-chips**, Elsevier, 2005.
7. L. Cheng, S. Hailes, "On-body wireless inertial sensing foot control applications", Proceedings of "PIMRC'08" IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, Cannes (France) 15-18 September 2008, pp. 1-5.
8. M. D. Cooney, C. Becker-Asano, T. Kanda, A. Alissandrakis, H. Ishiguro, "Full-body gesture recognition using inertial sensors for playful interaction with small humanoid robot", Proceedings of "IROS'10" IEEE/RSJ International Conference on Intelligent Robots and Systems*,* Taipei (Taiwan) 18-22 October 2010, pp. 2276-2282.
9. Q. Li, J. A. Stankovic, M. A. Hanson, A. T. Barth, J. Lach, G. Zhou, "accurate fast fall detection using gyroscopes and accelerometer-derived posture information", Proceedings of "BSN'09" the 6th International Workshop on Wearable and Implantable Body Sensor Networks, Berkeley CA (USA) 3-5 June 2009, pp. 138-143.
10. J. Skaloud, "Direct georeferencing in aerial photogrammetric mapping", *Photogrammetric Engineering and Remote Sensing*, Vol. 68, 2002, pp. 209-210.
11. S. Z. Jamal, "Tightly coupled GPS/INS airborne navigation system", *Aerospace and Electronic Systems Magazine*, Vol. 27, 2012, pp. 39-42.
12. H. Che, P. Liu, F. Zhang, Q. Wang, "A Deeply coupled GPS/INS Integrated navigation system suitable for high dynamic environments", Proceedings of "CSNC'12" the 3rd China Satellite Navigation Conference, Guanzhou (China) 15-19 May 2012, pp. 617-626.
13. S. Godha, G. Lachapelle, M. E. Cannon, "Integrated GPS/INS system for pedestrial navigation in a signal degraded environment, Proceeding of "ION GNSS'06" the 19th International Technical Meeting of the Institute of Navigation Satellite Division, Fort Worth TX (USA) 26-29 September 2006.
14. Y. Li, M. Efatmaneshnik, A. G. Dempster, “Attitude determination by integration of MEMS inertial sensors and GPS for autonomous agriculture applications”, *GPS Solutions*, Vol. 16, no. 1, 2012, pp. 41-52.
15. S. M. Warner, T. O. Koch, T. Pfau, "Inertial sensors for assessment of back movement in horses during locomotion over ground", *Equine Veterinary Journal*, Vol. 42, 2010, pp. 417-424.

16. G. Artese, V. Achilli, A. Trecroci, M. Gencarelli, G. Borgese, C. Pace, "Integrazione di strumentazione inerziale e GPS con fotocamera per fotogrammetria diretta: realizzazione di un prototipo e primi test", atti del convegno nazionale ASITA 2010, Brescia (Italy) 9-12 November 2010.
17. R. Dorobantu, B. Zebhauser, "Field Evaluation of a Low-Cost Strapdown IMU by means GPS", Ortung und Navigation, 1/1999, DGON, Bonn.
18. Oliver J. Woodman, "An introduction to inertial navigation", University of Cambridge, August 2007.
19. Z. F. Syed, P. Aggarwal, C. Goodall, X. Niu, N. El-Sheimy, "A new multi-position calibration method for MEMS inertial navigation systems", *Measurements Science and Technology*, No. 18, pp. 1897-1907, 2007.
20. G. Artese, M. Gencarelli, A. Trecroci, G. Borgese, C. Pace, "Metodologie di calibrazione delle strumentazioni inerziali: il modified multi-position calibration method per la calibrazione dei giroscopi", *Bollettino SIFET* (ISSN 1721-971X) , No. 1, pp. 171-83, 2010.
21. D.H. Titterton, J.L. Weston, "Strapdown Inertial Navigation Technology", Paul Zarchan Editor in Chief, second edition, 2007.
22. Analog Devices, "ADIS16350/ADIS16355- High precision tri-axis inertial sensor", datasheet rev. B, 2007-2009.
23. G. Artese, A. Trecroci, "Calibration of a low cost MEMS INS sensor for an integrated navigation system", Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2008 Beijing.
24. G. Borgese, L. Rizzo, C. Pace, G. Artese, "Compact Wireless GPS/Inertial System", Proceedings of "FBW'11" Fly by Wireless Workshop (FBW) the 4th Annual Caneus, Montreal (Canada) 14-17 June 2011.
25. L. Evans, P. Bryant, LHC machine, *Journal of Instrumentation*, vol. 3, 2008.
26. G. Bianchi et al, "FF-LYNX: fast and flexible electrical links for data acquisition and distribution of timing, trigger and control signals in future high energy physics experiments", *Nuclear Instruments and Methods in Physics Research A*, vol. 617 (2010) 289–290.
27. T. Grötker et al., **System design with SystemC**, Springer, 2002, ISBN 978-1-4020-7072-3.
28. Altera Corporation, Stratix II GX device handbook, Volume 1 and 2, 2009.
29. PLDA, XpressGXII reference manual, version 1.1, may 2008.
30. A. Ciampa, Grid data center INFN Pisa, Proc. of Workshop CEP 50, Pisa (Italy) November 2011.
31. K. Valasoulis, D. I. Fotiadis, I. E. Lagaris, A. Likas, "Solving differential equations with neural networks implementations on a DSP platform", Proceedings of the 14th International Conference on Digital Signal Processing, Santorini (Greece) July 2002.
32. L. Piñuel, I. Martin, F. Tirado, "A special-purpose parallel computer for solving partial differential equations", Proceedings of "PDP'98" the16th Euromicro Workshop on Parallel and Distributed Processing, Madrid

(Spain) 21-23 January 1998.
33. Y. Osana et all, "ReCSiP: An FPGA-based general-purpose biochemical simulator", *Electronics and Communications in Japan*, Part 2, Vol. 90, No. 7, 2007.
34. C. Huang, F. Vahid, T. Givargis, "A custom FPGA processor for physical model ordinary differential equation solving", *IEEE Embedded Systems Letters*, Vol. 3, No. 4, December 2011.
35. K. He, Y. Jiang, S. Dong, "A hybrid parallel framework for cellular potts model simulations", Proceedings of the 15th International Conference on Parallel and Distributed Systems, Shenzhen, Guangdong (China) 11 December 2009.
36. J. Hertz, R. G. Palmer,A. S. Krogh, **Introduction to the theory of neural computation**, Perseus Books, 1990, ISBN 0-201-51560-1.
37. L. O. Chua, L. Yang," Cellular neural networks: theory", *IEEE Transactions on Circuits and Systems*, 1998, 35:1257–1272.
38. L. O. Chua, L. Yang," Cellular neural networks: applications", *IEEE Transactions on Circuits and Systems*, 1998, 35:1273–1290.
39. T. Roska, L. O. Chua, "The CNN universal machine: an analogic array computer", *IEEE Transactions on Circuits and Systems II*, 1993, 40(3): 163-173.
40. P. Arena, L. Fortuna, A. Rizzo, M. G. Xibilia, "Extending the CNN paradigm to approximate chaotic systems with multivariable nonlinearities", Proceedings of "ISCAS 2000" the IEEE International Symposium on Circuits and Systems, Geneve (Switzerland) 28-31 May 2000.
41. L. Fortuna, A. Rizzo, M. G. Xibilia, "Modeling complex dynamics via extended PWL-based CNNS", *International Journal of Bifurcation and Chaos*, Vol. 13, No. 11, 2003, pp. 3273-3286.
42. O. Y. H. Cheung, P. H. W. Leong, E. K. C. Tsang, B. E. Shi, "A scalable FPGA implementation of cellular neural networks for gabor-type filtering", International Joint Conference on Neural networks, Vancouver, BC, (Canada) 16-21 July 2006.
43. B. G. Soos, A. Rak, J. Veres, G. Cserey, "GPU powered CNN simulator (SIMCNN) with graphical flow based programmability", Proceeding of "CNNA'08" the 11th International Workshop on Cellular Neural Networks and Their Applications, Santiago de Compostela (Spain) 14-16 July 2008.
44. R. Dolan, G. DeSouza, "GPU-based simulation of cellular neural networks for image processing", Proceedings of "IJCNN'09" the International Joint Conference on Neural Networks, Atlanta Georgia (USA) 14-19 June 2009.
45. M. Griebel, P. Zaspel, "A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations", *Computer Science-Research and Development*, Vol. 25, Issue: 1-2, May 2010
46. T. Y. Ho, P. M. Lam, C. S. Leung, "Parallelization of cellular neural networks on GPU", *Pattern Recognition*, Vol. 41, Issue: 18, August 2008.

47. Z. Nagy, P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA", *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 50, No. 6, June 2003.
48. Z. Nagy, Z. Vöröshazi, P. Szolgay, "Emulated digital CNN-UM solution of partial differential equations", *International Journal of Circuit Theory and Applications*, 2006, 34:445-470.
49. Z. Vöröshazi, A. Kiss, Z. Nagy, P. Szolgay, "FPGA based emulated-digital CNN-UM implementation with GAPU", Proceedings of the 11th International Workshop on Cellular Neural Networks and their Applications, Santiago de Compostela (Spain) 14-16 July 2008.
50. S. Kocsardi, Z. Nagy, A. Csik, P. Szolgay, "Two-dimension compressible flow simulation on emulated digital CNN-UM", Proceedings of " CNNA'08" the 11th International Workshop on Cellular Neural Networks and their Applications, Santiago de Compostela, (Spain) 14-16 July 2008.
51. D. J. Korteweg, G. de Vries, "On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves", *Philosophical Magazine*, 1895, 39:422–443.
52. Bilotta, E., Stranges, F. Pantano, P., "A Gallery of Chua attractors: Part III", *International Journal of Bifurcation and Chaos*, 17 (3), 657-734, 2007.
53. Bilotta, E., Di Blasi, G., Stranges, F., Pantano, P., "A gallery of Chua attractors. Part VI", *International Journal of Bifurcation and Chaos*, 17 (6), 1801-1910, 2007.
54. Bilotta, E., Pantano, P., "Emergent patterning phenomena in 2D cellular automata", *Artificial Life*, 11 (3) , pp. 339-362, 2005.
55. Bilotta, E., Pantano, P., "Structural and functional growth in self-reproducing cellular automata", Complexity, 11(6), 12-29, 2006.
56. Bilotta, E., Pantano, P. "The language of chaos", *International Journal of Bifurcation and Chaos*, 16 (3), 523-557, 2006.
57. M. W. Hirsch, S. Smale and R. Devaney, "Differential equations, dynamical systems, and an introduction to chaos", *Academic Press*, 2003. ISBN 0-12-349703-5.
58. P. G. Drazin, R. S. Johnson, "Solitons: an introduction", *Cambridge University Press.*, 2nd ed., 1989, ISBN 0-521-33655-4.
59. N. J. Zabusky, M. D. Kruskal, “Interaction of solitons in a collitionless plasma and the recurrence of initial states”, *Physical Review Letters*, 1965, 15(6):240-242.
60. W. Hereman, "Shallow Water Waves and Solitary Waves", *Encyclopedia of Complexity and Systems Science*, Springer, 2009, 8112-25.
61. H. Washimi, T. Taniuti, "Propagation of ion acoustic solitary waves of small amplitude", *Phys Rev Lett*, 1966, 17:996-8.
62. S. Giambò, P. Pantano, "Three-Dimensional Ion-Acoustic Waves in a Collisionless Plasma", *Lettere al Nuovo Cimento*, 1982, 34:380-4.
63. M. Wadati, " Wave Propagation in non linear lattice", *J. of the Phys Soc Jpn*, 1975, 38:673-80.

64. K. Fukushima, M. Wadati, T. Kotera, K. Sawada, Y. Narahara, "Experimental and theoretical study of the recurrence phenomena in nonlinear transmission line", *J Phys Soc Jpn*, 1980, 48:1029-35.
65. P. Pantano, "Inhomogeneous Dispersive and Dissipative Nonlinear Transmission Lines and Solitons", *Lettere al Nuovo Cimento*, 1983, 8:209-14.
66. G. Gallavotti (Ed.), "The Fermi-Pasta-Ulam Problem: A Status Report", *Lect. Notes Phys*, 2008, 728.
67. C. S. Gardner, J. M. Greene, M. D. Kruskal, R. M. Miura, "Method for solving the Korteweg de Vries equation", *Phys Rev Lett*, 1967, 19:1095-7.
68. C. S. Gardner, J. M. Greene, M. D. Kruskal, R. M. Miura, "The Korteweg de Vries equation and generalizations VI. Methods for exact solution", *Commun Pure ApplMath*, 1974, 27:97-133.
69. A. C. Vliegenthart, “On finite- difference methods for the Korteweg-de Vries Equation”, Journal *of Engineering Mathematics*,1971, 5(2):137-155.
70. L. Fortuna, M. Frasca, A. Rizzo, “Generating solitons in lattices of nonlinear circuits”, ISCAS 2001, The 2001 IEEE International Symposium on Circuits and Systems, pag. 680-683, vol. 2, May 6-9 2001, Sydney, NSW, Australia.
71. M. Remoissenet, **Waves called solitons**, Springer, 1996.
72. B. Luitel, G. K. Venayagamoorthy, "Decentralized Asynchronous Learning in Cellular Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, Vol. 23, Issue: 11, 2012.
73. M. Papadonikolakis, C. Bouganis, "Novel Cascade FPGA Accelerator for Support Vector Machines Classification", *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, Vol. 23, Issue: 7, 2012.