POLITECNICO DI TORINO

Master of Science in Electronic Engineering

Master Thesis

FPGA integration of a compressing system for satellite applications

Supervisors
prof. Maurizio Martina
prof. Enrico Magli

Candidate
Marco Cornelio

December 2019

Compiled with LaTeX on December 2, 2019.

Ringraziamenti

Ringrazio innanzitutto il professor Maurizio Martina che durante i vari corsi di studio mi ha fatto appassionare al mondo dell'elettronica digitale, portandomi per questo a sceglierlo come relatore del mio lavoro di tesi. Inoltre, mi ha dato la possibilità di partecipare a un progetto europeo in ambito aereospaziale, di cui ho sempre nutrito un notevole interesse. Infine lo ringrazio per essere stato sempre presente e a disponibile durante questi mesi di lavoro.

Un altro ringraziamento va dedicato al professor Enrico Magli e al professor Tiziano Bianchi, sempre pronti a supportarmi, qualora sorgesse qualsiasi dubbio sul codice da implementare.

Non potrei non ringraziare tutti i ragazzi del laboratorio del VLSI, che mi hanno aiutato con le loro competenze a risolvere problemi relativi ad argomenti per me sconosciuti, in quanto mai affrontati in nessun corso durante il mio percorso di studi. Oltre a questo sono stati degli ottimi compagni di laboratorio e hanno reso produttivi e molto più leggeri i giorni più difficoltosi.

Uno speciale ringraziamento va ai miei compagni di corso dell'università e agli amici di sempre, che mi hanno aiutato a concentrarmi e mi hanno sempre spronato ad andare avanti e a concludere il mio percorso di studi.

Devo ringraziare mio papà e mia mamma, che hanno sempre fatto tutto il possibile per far sì che io potessi concludere il mio percorso di studi e mi hanno sempre supportato in ogni momento di sconforto, incoraggiandomi a non mollare.

Infine ringrazio la mia ragazza Giorgia, che in questi anni è stata sempre presente ed, essendo anche lei un ingegnere, ha sempre compreso la fatica per raggiungere il fatidico traguardo. Un grande aiuto mi è stato dato durante la stesura di questo documento ed è soprattutto grazie a lei che sono riuscito a riportare nel modo corretto e nel tempo richiesto tutto il lavoro conseguito.

Summary

Earth observation is the use of a satellite, capable to scan the earth turning around the orbit in order to acquire information retrieving images. Nowadays Earth observation is considered a fundamental tool for many applications. In fact today's data retrieve from satellites analyses, for no-military use, is involved in different areas of application, for instance environmental monitoring, meteorology and map making. Its importance can be also confirmed by the big amount of financial resources invested in the sector. When some natural disasters occur, an additional aid may be provided by the implication of satellite images, which may be useful to analyse the situation and to supply suggestions to resolve the problem. One example of this may be the huge fire in Siberia and in the Amazon forest occurred in summer 2019, where satellite images were used to identify pyres that were already burning in order to extinguish them.

The EO-ALERT project is an H2020 European Union research activity coordinated by Deimos Space, started in January 2018 and lasting three years. The aim of this project is to build a next gen satellite architecture, changing the EO data chain to reduce latency in images production and delivery, reaching higher performances.

The extreme weather monitoring and ship detection are two possible fields of application for this type of satellite. In the former, information about meteorological phenomena can support more precise forecasts, allowing quicker alerts for extreme events like earthquakes, tsunamis and convective storms. In addition, information about surface winds can be useful for offshore wind farms and for hurricanes monitoring.

The latter instead may be used for coastal monitoring, focusing on maritime safety or illegal immigration. Moreover, also ship control as cargo monitoring and hampering illegal activity, such as fishing in conservation area or drug trade, may be another useful field of application.

Over the past 50 years, the approach used was to take earth images and to send them to ground for the post processing phase. The proposed chain implies images acquisition and a direct processing of them on board, sending elaborated images directly to ground. The system has to acquire SAR and optical images, processing, compressing and encrypting them before sending to the ground control. The goal of this process is to reach a latency less than or equal to 15 minutes for the image processing, starting from the end of sensor acquisition to the receiving time at ground control.

The best solution to perform on board processing is to use a processor with FPGA support, on which an hardware accelerator should be implemented.

The partners of the project are Deimos Space, DLR, Technische Universität Graz, Politecnico di Torino, OHB Italia and Deimos Imaging. Each partner has to work on a different subsystem of the satellite architecture and has at disposition

a Zynq ultrascale+ MPSOC ZU19EG, which has lots of space on the FPGA and two processors: ARM Cortex-A53 with 1.5 GHz clock and ARM Cortex-R5 with 600 MHz. This board has been chosen considering the resources needed by the processing subsystem, which is the most resource-consuming. This board is composed by a motherboard which allows to connect the Zynq board with some extension boards, in order to increase the available resources. The whole system is composed of several Zynq boards, each of them representing a different satellite subsystem. Each board has at disposition many components. The memory part is provided by two DDR4 memories, for a total of 9 Gbytes, allowing a proper elaboration of the image. In addition three extension boards are implied, in order to increase the mutual communication with other boards: one with two Gbit Ethernet ports, another with two QSFP connectors on which a PCIe communication is implemented on and a final one with another Gbit Ethernet and SATA ports. In particular the last one is useful for storing processed and compressed images before sending them to ground.

Politecnico was assigned the compression and encryption parts. The task to perform is related to the compression code to implement on the board, then converting most of it into hardware.

The first part of the thesis is about connecting and running the Zynq board with all the extension boards plugged. This part was focused on understanding how to use and program the board, in order to define an hardware architecture to run the compression code on.

Firstly the motherboard was programmed with the aim of establishing a comunication between the FPGA and the processor with all the extension boards. Using the tool provided, it is possible to generate the pin mapping file, useful to connect the extension boards to the motherboard, and in addition to provide power supply to each part of the system without exceeding the recommended limits.

Once this part has been performed, each extension board was configured and debugged. An OS was also created on the hardware description, in order to facilitate the device control.

After the hardware architecture has been defined, all the C code was compiled in board architecture and tested on it in order to check the compatibility. This part may be considered as a backup of the project, in case of some issues occurred during hardware implementation.

The second part of the thesis is about converting the C code of the compressor into VHDL code, so that it can be loaded on the FPGA, allowing the processor to perform other tasks at the same time while the compression is executed on the hardware.

Compression of optical and SAR images is based on the CCSDS lossless and near-lossless compression standard for space data system. This standard is based on Differential Pulse Code Modulation (DPCM) prediction loop that, using a spatial/spectral predictor, can compute current pixel to encode. This may be possible

with a function which involves some previous encoded neighbouring samples of the spectral channel and some of the previous encoded spectral channels. Once the preceding steps are performed, the prediction residuals are encoded and then encrypted using Keccak functions.

The Vivado synthesizer it is not able to convert the previous code from C language to VHDL, due to some high level functions and the complexity of Keccak library. Since it is not considered wise to modify a certificated and validated library, it has been chosen to modify the VHDL code provided by Keccak authors and to implement the required part by hand.

From the remaining part of the project, among all the blocks, it was decided to pass to the synthesizer only the predictor function. Such a choice was taken in order to at least synthesize a reduced part of the compressor, since the synthesizer is still not able to understand the whole code.

After simplifying some functions, a synthesizable version of the predictor was obtained.

This caption represents the current work state. Additional efforts have to be done in order to complete the conversion of all the compressor blocks to assemble, also including the Keccak hand made part. This study outlines the Zynq ultrascale+ functions in all their parts and complexity. At the state of art this thesis represents a crucial starting point for the comprehension of this family board, hoping it will pave the way for completing the current project state.

Contents

List of Tables 11

List of Figures 13

List of acronyms and abbreviations 17

1 Introduction, Reasons and Goal 20
- 1.1 Introduction on EO-ALERT project . . . 20
- 1.2 Purpose of this thesis . . . 26
- 1.3 Thesis outlines . . . 26

2 Overview on EO-ALERT architecture, compression/encryption code and FPGA board 27
- 2.1 Avionic architecture . . . 27
- 2.2 Compression . . . 30
  - 2.2.1 Compression structure . . . 31
  - 2.2.2 Compression code . . . 34
- 2.3 Encryption and Keccak algorithm . . . 37
  - 2.3.1 The sponge construction . . . 38
  - 2.3.2 Keccak family sponge functions . . . 39
  - 2.3.3 Padding rule . . . 44
  - 2.3.4 Duplex sponge and PRGsponge . . . 45
- 2.4 Hardware equipments . . . 49
  - 2.4.1 Zynq Ultrascale+ MPSoC . . . 49
  - 2.4.2 proFPGA motherboard . . . 51
  - 2.4.3 Extension boards . . . 52

3 Tools programs 55
- 3.1 ProFPGA builder . . . 55
- 3.2 Vivado . . . 58
  - 3.2.1 Problems . . . 60
- 3.3 Vivado SDK . . . 61
  - 3.3.1 Problems . . . 63

3.4 PetaLinux 63

4 Hardware structure 69
4.1 Base tests : FPGA (PL), processor (PS) and together (PS + PL) . 69
4.2 Test of the interrupt functions 72
4.3 GBit Ethernet Board (EB-PDS-GBITETHERNET-R1) 72
4.4 DDR4 5 Gbytes ram Board (EB-PDS-DDR4-R6) 76
4.5 Custom AXI slave VHDL implementation controlled by OS 77

5 Code implementation 81
5.1 Keccak C code analysis 82
5.2 VHDL Keccak implementation 83
5.2.1 Compression code implementation 91

6 Conclusions and future work 93

Appendix A Configuration file generated by ProFPGA 95

Appendix B Script for automatic generation 101
B.1 Tcl file that instantiate Zynq IP block in Vivado block diagram . . 101
B.2 Bash for petalinux to create OS system for the Zynq board 103

Appendix C Base tests 107
C.1 Only FPGA test files 107
C.2 Processor and FPGA first design 109
C.3 Processor and FPGA second design 112
C.4 Test of processor interrupts 122
C.5 Test of Gbit Ethernet 128
C.6 Test of DDR4 136
C.7 Test of Custom slave device 143

Appendix D Configuration file generated by ProFPGA 161

Bibliography 169

List of Tables

2.1 Rho rotation offset corresponding to X and Y coordinates with w=64. 42
2.2 Round constant used for Iota function with w=64. 44
2.3 Preliminary evaluation of used resources and comparison with available resources on ZCU106 board. 49
2.4 Available resources on ZU19EG board 49

List of Figures

1.1 Difference of EO data chain for EO image production: (left) classical data chain based on raw data compression and transfer, (right) innovative data chain showing its key elements and new data flows (in red). 22
1.2 EO-ALERT architecture. 24

2.1 EO-ALERT system architecture updated. 28
2.2 Compression and encryption block diagram. 29
2.3 Block diagram of prediction-based near-lossless compression. 32
2.4 Coordinates system used for input samples. 33
2.5 Coordinates system used for input samples. 33
2.6 Basic concept of hashing function. 37
2.7 Sponge construction diagram. 38
2.8 State 3D vector of Keccak function. 40
2.9 Definition of smallest block of state array with an example with w=8. / Keccak-images 40
2.10 Application of Theta round on the state vector with w=8. / Keccak-images 41
2.11 Application of Rho round on the state vector with w=8. / Keccak-images 42
2.12 Application of Pi round on the state vector with w=8. / Keccak-images 43
2.13 Application of Chi round on the state vector with w=8. / Keccak-images 43
2.14 a) Padding rule 10*1 applied when the input message is equal to r bits and b) when it is less than r bits, in case of little-endian and big-endian. 45
2.15 The Duplex construction. 46
2.16 Generating the output of a duplexing call with a sponge. 47
2.17 Zynq UltraScale+ MPSoCs: EG Block Diagram 50
2.18 Example of proFPGA modular hardware approach system. 51
2.19 ProFPGA uno motherboard. 52

2.20 Scheme of the coordinate system used on the board to identify extension boards plugged. 53

3.1 ProFPGA builder graphic environment with all the boards connected with their details. 56
3.2 Functional block digram of the programmable clock generator SI5338. 57
3.3 Vivado GUI. 59
3.4 Vivado GUI showing block diagram. 59
3.5 Vivado tlc part in which the code and description of the error are reported. 61
3.6 Vivado Zynq UltraScale+ MPSoC IP block customization. 61
3.7 Vivado Zynq UltraScale+ MPSoC IP block customization. 62
3.8 Command line showing settings after petalinux–get-hw-description. 65
3.9 Command line showing settings after petalinux–get-hw-description. 66
3.10 Command line showing settings after petalinux–get-hw-description. 66

4.1 Gbit Ethernet board. 73
4.2 Diagram of interface communication between FPGA and Ethernet port with the three possible positions to add skew: 1)FPGA, 2)PCB and 3)PHY. 74
4.3 Gmii To Rgmii IP block design with relative possibility to add skew. 75

5.1 On the upside there is the keccak VHDL code, while in the bottom side there are the graphical representations of the taken block in the state vector in order: output,rate and capacity. Red blocks represent the first for cycle, while green the second. 85
5.2 Block diagram of Keccak block with all the components and signals used. In dark grey the input block (6a bits) and the output vector (256 bits) sections are highlighted. 87
5.3 ASM chart of the keccak test. 89
5.4 ASM chart of the keccak controller. 91
5.5 Times taken for having valid data at the output and for completing the reading all 1016 output bits. 92

C.1 Block diagram of the test_hello design. 111
C.2 Block diagram of the test_up_down design. 121
C.3 Block diagram of the test_interrupt design. 126
C.4 Block diagram of the test_client design. 131
C.5 Zynq Ip settings for the Ethernet port pins. 131
C.6 Test of the Ethernet with client running on the Zynq. 132
C.7 Test of the Ethernet with server running on the Raspberry. 132
C.8 Test of the Ethernet with server running on the Zynq. 132
C.9 Test of the Ethernet with client running on the Raspberry. 133

C.10 Ping test from Zynq to raspberry. 133
C.11 Block diagram of the test_DDR design. 142
C.12 Block diagram of the test_calc design. 146
C.13 Test on OS of the multiplier with positive numbers. 159
C.14 Test on OS of the multiplier with negative numbers. 159

List of acronyms and abbreviations

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ASM	Algorithmic State Machine
AXI	Advanced eXtensible Interface
BIL	Band Interleaved by Line
BSQ	Band SeQuential
CAN	Controller Area Network
CCSDS	Consultative Committee for Space Data Systems
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access
DMBI	Device Message Box Interface
DPCM	Differential Pulse Code Modulation
ELF	Executable and Linkable Format
EO	Earth Observation
FDIR	Fault Detection, isolation and recovery
FPGA	Field Programmable Gate Array
FSBL	First Stage Boot Loader
FSM	Finite State Machine
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical User interface
IP	Intellectual property
IR	Infrared
LSB	Least Significant Bit
MAC	Media Access Control
MAC	Message Authentication Codes
MIG	Memory Interface Generator
MSB	Most Significant Bit
MSE	Mean Square Error
NIST	National Institute of Standards and Technology

NRT	Near Real-Time
OHB	Orbitale Hochtechnologie Bremen
OS	Operating System
PCIe	Peripheral Component Interconnect Express
PHY	PHYsical layer
PL	Programmable Logic
PLL	Phase Locked loop
PMUFW	Platform Management Unit FirmWare
PRG	Pseudo Random Generator
PS	Processing System
QSFP	Quad Small Form-Factor Pluggable
RAM	Random Access Memory
RGMII	Reduced Gigabit Media-Independent Interface
SAR	Synthetic Aperture Radar
SATA	Serial Advanced Technology Attachment
SD-card	Secure Digital card
SDK	Software Development Kit
SHA-3	Secure Hash Algorithm
SO-DIMM	Small Outline Dual In-line Memory Module
TCP	Transmission Control Protocol
TX/RX	Transmitter/Receiver
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VHDL	VHSIC Very high speed integrated circuits Hardware Descriptio Language

Chapter 1

Introduction, Reasons and Goal

1.1 Introduction on EO-ALERT project

The first occurrence of satellite remote sensing can be dated to the launch of the first artificial satellite, Sputnik 1, by the Soviet Union on October 4, 1957 [1]. Once sent this metal sphere in earth orbit, Russian scientists could not estimate the success of the mission would have. After reaching a low elliptic orbit around the world, the satellite stayed in trajectory for three weeks till the depletions of the battery and then, after two other weeks of uncontrolled flight, fell down to the earth. By tracking the position of Sputnik and by considering the resistance on the orbit, scientists were able to estimate the density of the atmosphere and to analyse the ionosphere for the first time, thanks to the study of the propagation of the radio signals emitted from the antennas. After this achievement, United States started their space program and the so called "space race" began between the two Cold War rivals. During this period huge improvements and discoveries were accomplished and the use of satellites became very widespread.

Nowadays Earth Observation (EO) is considered a fundamental tool for many applications. In fact today's data retrieve from satellites analyses, for no-military use, is involved in different areas of application, for instance environmental monitoring, meteorology and map making. For example, Hera company, that manages water distribution in some Italian regions, was able to detect and repair water losses over all the distribution with very high accuracy [2], thanks to satellite images that highlighted soaked fields with huge level of chlorine. During '80, after those infrared sensors were mounted on satellites,the hole in the ozone was detected, discovering the correlation between them and the emission of fridge and spray gases in the atmosphere. These are only some examples of the importance of satellites used in modern times.

There are two new applications of satellites that may respectively create two new businesses. The former was developed by Sarah Parcak, an American archaeologist who uses satellites infrared images to identify potential archaeological sites; in fact

clay bricks absorb the water under the soil and leave a difference trace in the infrared images. This is also the same method used by spacecraft Mars Express in 2018 to detect water liquid presence under Mars's glaciers. This can be considered as the birth of "space archaeology". The second new application was provided by the American Space-Know start-up and it is about the financial sector. For many years, analysts had difficulties in studying the Chinese's economy growth due to the incomplete documents provided by the industries, till when this start-up found a new solution. Indeed, it started to evaluate with satellites imaging the variation of goods stocking, trucks movements, roads and warehouse size changes. These measurements became so accurate to be part of an official index of Bloomberg, known as Satellite Manufacturing Index (Smi).

Over the past 50 years, the EO data chain consisted in the acquisition of data from sensors of the satellite, compression, storage on-board, sending to ground for processing and creation of the EO image. Due to the huge quantity of data, their transfer required significant amount of time. With the increasing demand, more responsivity and accuracy are required, with consequent increase of data management in less time. In this condition, the classical EO data chain generates a severe bottleneck problem: the more large the data obtained from sensors (raw data), although its compression, the more slow the EO product availability becomes, increasing latency and hampering applications to grow in accordance with the increased user demand for EO products. The EO-ALERT project [3], an H2020 European Union research activity, addresses the challenge of a "high speed data chain" and the need for increased EO data chain throughput.

In order to obtain these results, a combination of innovations in the on-board elements of the data chain and in the communications link is needed. The main possibilities provided are on-board reconfigurable data handling, on-board image generation, on-board image processing, high-speed on-board avionics, on-board data compression and reconfigurable high data rate communication links to ground. These innovations also provide a possible optimisation of the classical EO data chain towards a data chain with greatly improved data throughput.

Data latency is one of the most important parameters used to characterize performances of EO chain. In fact for a useful system, final users require data with low errors available in a very short time interval. Some examples are reported in order to make performances comparison with already known working systems.

In case of polar platform satellites, the provision of image products is in the range of 1 to 3 hours, known as near real-time Near Real-Time (NRT); for instance, Sentinel-1 makes ocean products available within 1 hour of observation over NRT areas. Nowadays technology latency is going beyond NRT applications with latencies in the range of 30 minutes to 15 minutes. In order to go ahead in performances, the concept behind EO-ALERT is to achieve latencies below 15 minutes for the EO products.

In the EO chain, latency can be defined as the interval between the time of

the collection of the last photons and the time in which the data is all converted to a specified EO product and delivered to the user portal. In particular with this latency definition, EO-ALERT has the aim of reaching a maximum latency of less than 5 minutes, for both Synthetic Aperture Radar (SAR) and optical image products.

As mentioned before, usually data handling is characterized by the collection of raw data, and then by the compression and the scheduling for the transmission to the ground segment, where data processing takes place. This flow of image production requires lots of time, and due to the significant delay in the generation, this can introduce a wrong representation of the acquired image. In fact if the image is available after long time, it may represent old scan conditions, not more useful for real-time analysis.

With new powerful computational systems for onboard data handling, it is possible to change the processing chain. The most relevant improvement consists in the direct generation of data products, from raw data, on-board, reducing the time needed to generate them, from hours to a few minutes. In figure 1.1 improvements to data chain are illustrated and the difference with classical flow is highlighted.

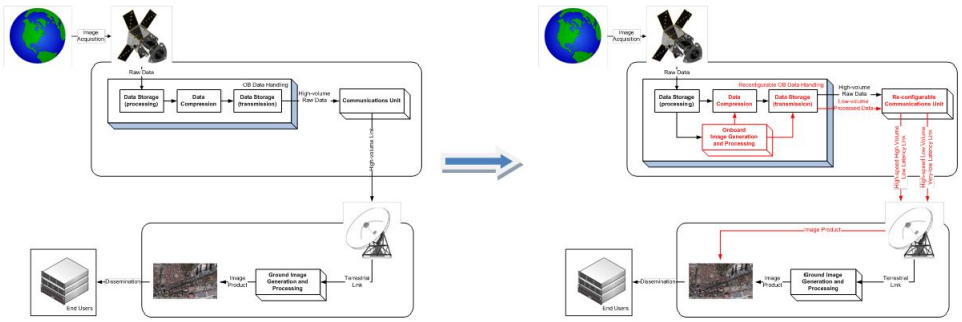

Figure 1.1. Difference of EO data chain for EO image production: (left) classical data chain based on raw data compression and transfer, (right) innovative data chain showing its key elements and new data flows (in red).

In order to evaluate the performances of the new data chain, EO-ALERT is tested on two different environments: ship detection and extreme weather monitoring, that are briefly described below.

Ship detection

In ship detection the most important aspects are the reduced size of the pixel, at least 1m, and the low revisit time. With these requirements SAR satellites become the most suitable for ship detection. Moreover thanks to very high resolution

of optical imagery, false positive are reduced and ship detection is allowed. In particular, two tests are proposed for Mediterranean sea:

- Ship Monitoring: the objective is focused on illegal fishing and cargo monitoring between 10-20 meters.
- Coastal Monitoring: the objective is focused on maritime safety and illegal migration.

The generated EO products should be a text file, in which the following information is included:

- Position Information;
- Movement Information;
- Ship details;
- Ship identification;
- Clipping ship.

The ship detection is divided in two steps: coarse detection and discrimination. The coarse detection consists in limiting the searching area instead of analysing all the image, by reducing the number of ships considered at the same time. This approach allows to divide in smaller subsets all the ships to be analysed, reducing the complexity and, at the same time, decreasing false positive detection caused by the presence of land in the image. This part uses the Otsu's method, an algorithm that finds an optimum threshold value in order to define two different classes. Employing this technique it is possible to separate ships from land; in fact water presents a low reflectivity compared with land and clouds, allowing fast water non-water segmentation.

Then each subset is analysed in the discrimination step, in which ships are separated from false positive detection, in order to generate correct information about the image. If the target is recognized as a ship, it can be described by a set of features designed to encode its gradient distribution, by analysing the direction of the edges inside a pixel neighbourhood.

Extreme weather

The other scenario where EO-ALERT can be very useful is meteorology. Information about meteorological phenomena can support more precise forecast and it can allow quicker alerts for extreme event like earthquake,tsunami and convective storms. In addition, information about surface winds can be useful for offshore wind farms and monitoring of hurricanes.

Two meteorological phenomena are going to be detected in this scenario: convective storms and surface ocean winds and overseas. In case of convective storms, four stages of convection should be detected: pre-convective environments, convection initiation, mature and dissipation stages of the storms. Data get from SAR are used for surface wind speed estimations overseas and oceans, in order to perform storm detection and monitoring, which may be very useful in providing specific information on individual storm location, trajectory and characteristics.

For extreme weather detection, Infrared (IR) images will be used together with optical images, in order to retrieve the information on the cloud top temperature. The first step of the image process is to detect some cold cells by using the same approach of ship detection: all the areas colder than a threshold are marked as possible storms. Then, the identified cold cells are tracked over time by analysing visual features, extracted from the optical images, and the temperature profile, extracted from IR images.

EO-ALERT architecture

The aim of EO-ALERT project is to define a flexible data-handling architecture, managing both optical and SAR data, in order to use on-board resources for providing high-speed data acquisition, processing, and generation of rapid alerts.

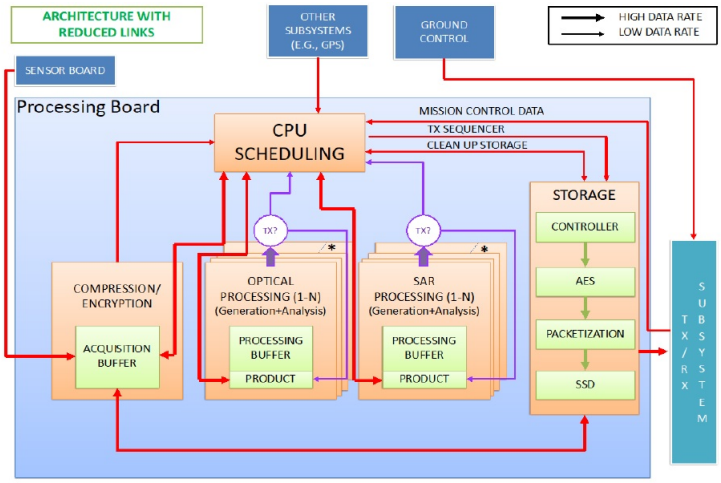

Figure 1.2. EO-ALERT architecture.

In figure 1.2 is described the proposed architecture, designed following three key points:

- Modular: the architecture is divided into blocks, each with a defined function. In this way every section is divided by the others and any modification to a block or function has a minimal or no impact on the others;

- Scalable: the architecture can be adapted to different data types, e.g. optical/SAR, and to different sensor types for each kind of data (e.g., images from different sensors or having different sizes), within the maximum memory and computational capabilities provided by the avionics;
- Reconfigurable: the architecture allows to modify the computing resources for the optical and SAR processing functionalities. Reconfiguration can be done on-the-fly through download of new software from the Central Processing Unit (CPU) to the board that has to change the software. With this approach it is also possible to provide Fault Detection, isolation and recovery (FDIR): if a board has an irreparable failure, it can be possible to reconfigure the software and to perform the previous tasks in another board, avoiding the problems due to the failure. This method is very innovative and convenient, with respect to conventional ones employing redundant boards. In this way the cost and weight can be reduced with a little effort in software design.

Based on the remarks above, the architecture was designed using three main blocks: the Sensor Board, the processing Board and the Transmitter/Receiver (TX/RX) subsystem.

The Sensor Board acquires the optical/SAR raw data and transfers it to the acquisition buffer.

The Processing Board consists of a CPU board and some Field-Programmable Gate Array (FPGA) boards, each one with its own Random Access Memory (RAM) and different functions, in particular they are dedicated to:

1. Compression/encryption;
2. Optical processing;
3. SAR processing;
4. Storage.

All other added boards are used to increase computational power for image processing. All boards are connected with the CPU with point-to-point data link, in order to realize data transfer and to allow reconfiguration. This type of links allows to avoid the use of shared buses, preventing possible congestions during the data transfer and guaranteeing a low latency. In the figure 1.2 connections are illustrated between the Sensor Board, the Processing Board, the TX/RX subsystem and all the internal connections in the Processing Board with arrows. As it is possible to notice, the star point of the system communication is the CPU. Moreover storage and compression/encryption tasks can be put on the same board, in order to reduce the number of boards involved or to reuse uninitialized boards for image processing, increasing the performance. All this information and more details can be found in the presentation paper of EO-ALERT project [4].

1.2 Purpose of this thesis

The purpose of the thesis is to build and to make the hardware system running. It is composed of motherboard, FPGA and extension boards, on which the compression code for satellite images will work. Once the system is completed, the aim is to implement the compression code on the FPGA as an hardware block. The implementation can be considered as an HW accelerator that the OS system, running on the board, can use without being stuck during the running of other programs. The target is to convert the compression code into VHDL language and, if it is possible, most of it, in order to achieve the best performances.

1.3 Thesis outlines

In order to better visualize the content of this thesis, the topics that will be analysed in the various chapters are reported below.

In chapter 2 an overview of the satellite architecture, a description of the compression code that has to be implemented on the board, a description of the encryption code and Keccak library and the hardware used with the board to realize the hardware system will be provided.

In chapter 3 instead, a list of the tools, used to develop the system configuration and the code implementation, will be described. For each of them the common steps for configuring and using the system will be given, moreover also the problems encountered are listed with corresponding solutions to solve them.

All the steps followed to make running the board and to connect all the extension boards in the correct way, by ensuring the proper functioning, will be reported in chapter 4. The description starts from the first test using only the FPGA or the processor for the creation of the OS to run on the board with an hardware accelerator.

In chapter 5 the encryption VHDL code implementation will be described, starting from the VHDL Keccak hash function.

At the end, the conclusions of the achieved goals and the future work are listed.

Chapter 2

Overview on EO-ALERT architecture, compression/encryption code and FPGA board

2.1 Avionic architecture

In the project paper [4] the architecture suggested has compression/encryption storage separated from the CPU scheduling and different type of PCIe communication links. After some reviews, a new architecture has been proposed by OHB in D3.8 document [5].

In figure 2.1 a scheme of the architecture is provided; in this version compression/encryption storage is put with CPU scheduling on the same board. The system consists on: CPU scheduling and compression/encryption storage, master optical processing, master SAR processing and spare processing board. As mentioned above, the system can be reconfigured in case of irreversible damage of the board, avoiding redundant boards, although a redundant system, equal to the main one, is used anyway in the architecture, in order to reduce risk of failure.

The CPU scheduling (**M**ain and **R**edundant) is used mainly for the data transfer management. The CPU is the centre of the star and can communicate with all the other boards. The raw data generated by SAR and the optical sensors are sent to the CPU scheduling board through TTethernet (grey arrow) and then forwarded respectively to the SAR and to the optical processing boards through Peripheral Component Interconnect Express (PCIe) Gen3 type (red arrows). In particular TTethernet is an Ethernet-based communication system, developed for time critical application and for managing data rates up to 1000Mbps. After the processing,

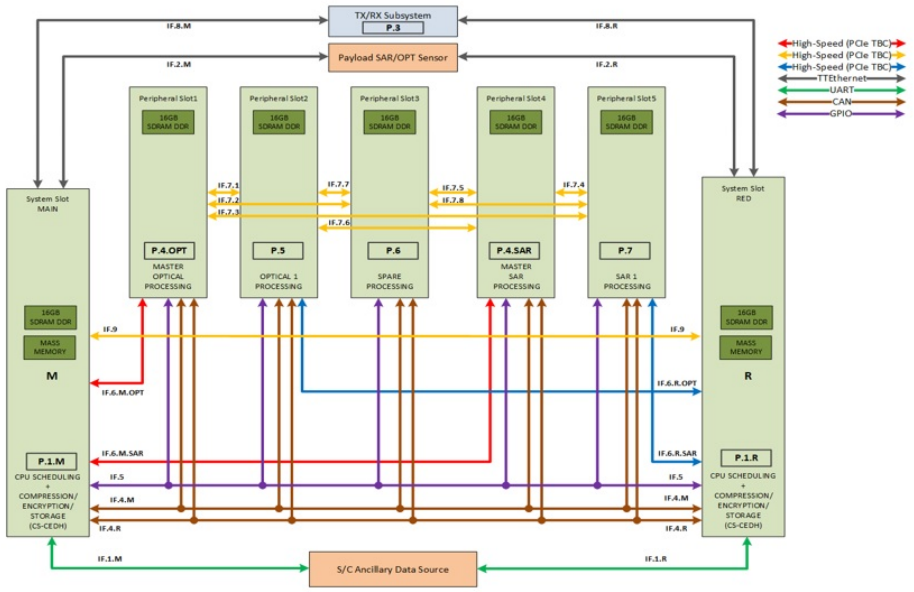

Figure 2.1. EO-ALERT system architecture updated.

the images produced return to the CPU board where they are compressed, encrypted and stored in the mass storage, forwarding relevant data to the TX/RX subsystem in order to transmit them to Earth. The CPU scheduling retrieves also ancillary data from the relevant subsystems (e.g., Global Positioning System,time), used for processing and rearranging images sequences. These communication links (green arrow) are universal asynchronous receiver-transmitter (UART) type. From TX/RX subsystem, through TTethernet, CPU scheduling obtains ground control data, parameters, configurations, and, generally, any information needed to execute properly a given mission. There is also a Controller Area Network (CAN) link (brown arrow), used as a redundant communication, that connects all the boards with the CPU scheduling for configuration, monitoring, telemetry/commands and low data rate link purposes. In addition, a set of General Purpose Input/Output (GPIO) (purple arrow) are exchanged between the boards for generic purposes. When the CPU scheduling sends image data to the SAR and to the optical processing boards, images can be also forwarded to additional processing boards in order to parallelize the calculations with subsequent improvement of the performances. For this reason a further PCIe link is used (yellow arrow) for the exchange of image/product data between processing boards. For instance Optical 1 Processing board is a redundant board used for the processing of the optical images and, in nominal condition (i.e. no failure), it receives image data from the master one for the parallelization of the algorithm. In case of master board failure, it is switched off and the subsystem is reconfigured such as the redundant board becomes the

new master optical processing board.

The Spare Processing board instead can be configured as both optical processing board and SAR processing board, depending on the application.
In particular, the thesis is focused on the implementation of the compression and on the encryption algorithm, thus an accurate description of the architecture is required.

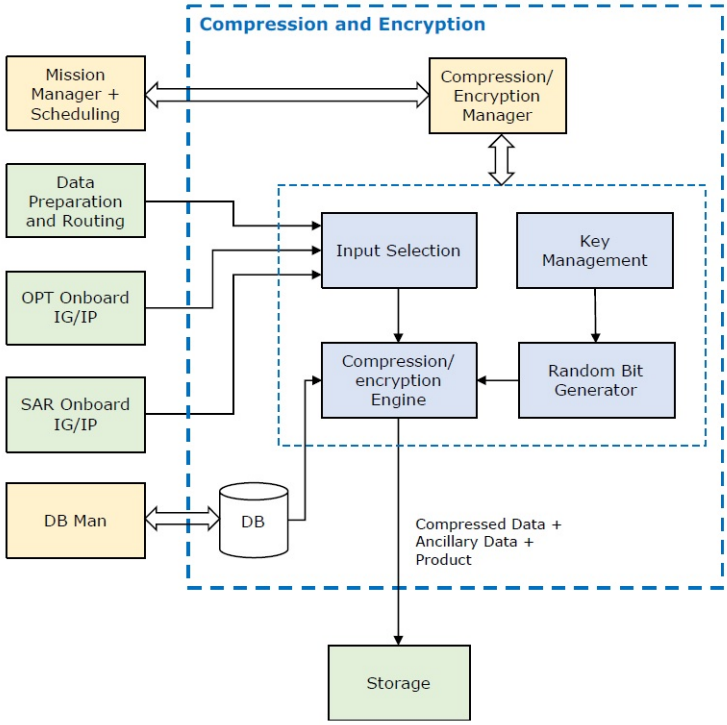

Figure 2.2. Compression and encryption block diagram.

In figure 2.2 a reference design for the compression and encryption board to implement is provided, obtained from the specific delivered in the data chain description [20].

The Compression/Encryption Block receives the raw data from the sensor board and it stores it in the acquisition buffer, which is defined in the RAM connected to the board and used to save intermediate calculations of the compression/encryption processes. In general the block has to do the following tasks :

- Inform the CPU about the receiving of raw data;
- Prepare the data for processing (e.g., tiling);
- Compress/Encrypt data;

- Store processed data in the mass storage;
- In case of alert, after compressing/encrypting the products generated by the optical processing board and the SAR processing board, forward them to the tx/rx subsystem in order to send them to ground.

In general, compression/encryption blocks have to work with all the data type available on the system, including both optical and SAR raw data, optical and SAR images products, generated after image processing, ancillary and Global Positioning System (GPS) data and finally images products and alerts that have to be send to ground. There is a mass storage, connected to the board through Serial Advanced Technology Attachment (SATA) connection, used to store the data to be forwarded to the TX/RX subsystem. Every image generated is compressed and sent to the mass storage, while images containing important information are also forwarded to the TX/RX board. In order to perform this task and use an efficient data-handling policy, a controller is implemented on the board. The controller receives information of date, time and GPS location from the CPU and through them it rearranges the data to be queued for transmission, removing from the storage also data that doesn't need to be transmitted.

The queuing order is based on "normal" and "high" priorities, typically associated to image data and product data respectively. High priority data may be transmitted first, causing the consequent interruption of normal priority data transmission in order to reduce latency.

2.2 Compression

Compression of optical and SAR images is based on the CCSDS lossless and near-lossless compression standard for space data system [6]. This standard is based on a Differential Pulse Code Modulation (DPCM) prediction loop that, using a spatial/spectral predictor, can compute current pixel to encode. This may be possible with a function which involves some previous encoded neighbouring samples of the spectral channel and some of the previous encoded spectral channels. In order to obtain the most accurate prediction, which is the smallest difference between current pixel and the predicted one, the prediction algorithm is adaptive. This means that for every pixel neighbouring coefficients are combined linearly and evaluated every time. After this prediction, residuals are quantized, while entropy is coded by using the Golomb coder. For encryption, two applications are used. One is used in the compression for the sign randomization algorithm of prediction residuals and for the randomization of the entropy encoder. In the second case, encryption is used to encrypt message alerts before sending them to receivers, without compression.

The term near-lossless compression refers to a kind of lossy compression, in which the maximum error between each original and reconstructed pixels is bounded

by some user defined parameters. Due to this constraints, it is possible to control the image quality and to avoid more serious errors that can affect the image interpretation during the reconstruction phase, like false detection or anomalies. In particular near-losses algorithm can be considered as a trade off between lossless and lossy compression, borrowing the high quality reconstruction from the former and the large compression ratio from the latter, which is impossible to reach in lossless.

CCSDS defines a standard for multicomponent images in lossy compression algorithms, such as multispectral and hyperspectral images [6]. These algorithms are defined for application to on-board satellite compression [7], where there are restrictive constraints in terms of memory, computational capability and available hardware.

Usually low encoder complexity is preferable, since on board system there are limited resources due to the power capability, available space and radiation hardening. Moreover, since in the project a FPGA is used, on which the algorithm is hardware mapped in VHDL, it is preferable to have an encoder with low complexity and an algorithm with easy operations to map to a VHDL description.

Generally in this type of algorithm it is preferable to send to earth a first low-quality version of the subimage of interest, allowing the user to carry out a first evaluation of the image, rejecting all the useless ones without wasting time, if communication problems allows a fast communication.

The near-losses algorithm have to minimize the Mean Square Error (MSE), reducing the bit rate during the compression process and bounding the maximum error during the reconstruction phase. In particular the user defines a parameter Δ, used to specify the maximum acceptable error between the original image and the decoded one such as the following condition is respected:

$$max_{i,j,k} \|x_{i,j,k} - \hat{x}_{i,j,k}\| \leq \Delta$$

where x is the original pixel and $\hat{x}$ is the reconstructed one. If Δ is chosen in a proper way, then the compression is very similar to a lossless one. Indeed if the maximum error introduced by compression is smaller than the noise, the image reconstructed has a quality very similar to an image reconstructed with lossless compression.

2.2.1 Compression structure

As aforementioned, the compressor can be schematized as in figure 2.3 on a DPCM prediction loop [9]. The compression is based on the spatial/spectral predictor, with which it is possible to estimate the value of the current pixel to encode. The more the predicted pixel is near to the correct value, the more the energy of the residual can be reduced. To obtain the most accurate prediction, the predictor is implemented with an adaptive mechanism, which changes for every pixel the

coefficients of the prediction. After the sample is predicted, this value is subtracted to the original one, using the sign algorithm. Indeed, it changes the coefficients considering the sign of the prediction residual, minimizing its value. The decoded values are the input of the predictor, instead of the original pixel. This allows the decoder to adapt the linear combination coefficients without passing them explicitly. After the predictor residual are evaluated, they are the only one to be quantized, while the entropy is encoded and written in the compressed file.

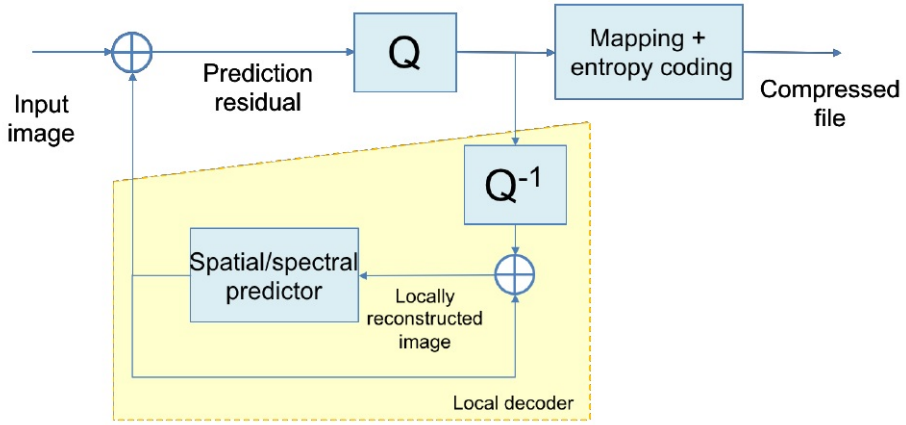

Figure 2.3. Block diagram of prediction-based near-lossless compression.

In the backward loop the quantized predictor residuals are dequantized and summed to the estimate value, obtaining the decoded samples, set as input of the local decoder. This construction allows to have, for both encoder and decoder, the same sequence of decoded inputs. Compression is performed in a single step, processing all the image that can be provided in two different scan orders: Band Interleaved by Line (BIL) and Band SeQuential (BSQ).

Samples used for the calculation are in the current and in the P previous spectral bands, where P is a user defined parameter that defines the number of bands used for prediction. Typically, this value is P=3, while bigger values are justified only when the spectral correlation is higher, like in case of atmospheric sounders. Prediction is performed using an adaptive linear prediction which updates the weight, according to the sign algorithm. Each pixel is defined using three coordinates in this way $S_{z,y,x}$ where x is the horizontal position in the image, y is the vertical one and z is related to the spectral band as show in figure 2.4. The input passed to the predictor is not the original image pixel, but the decoded sample, which is indicated as $S''_{z,y,x}$.

In order to remove the mean value of data, the "local sums θ" are computed for each pixel in the same spectral band. There are three different procedures to evaluate these sums: wide neighbor-oriented, narrow neighbor-oriented and column-oriented. A visual description of them is provided in figure 2.5. In particular the

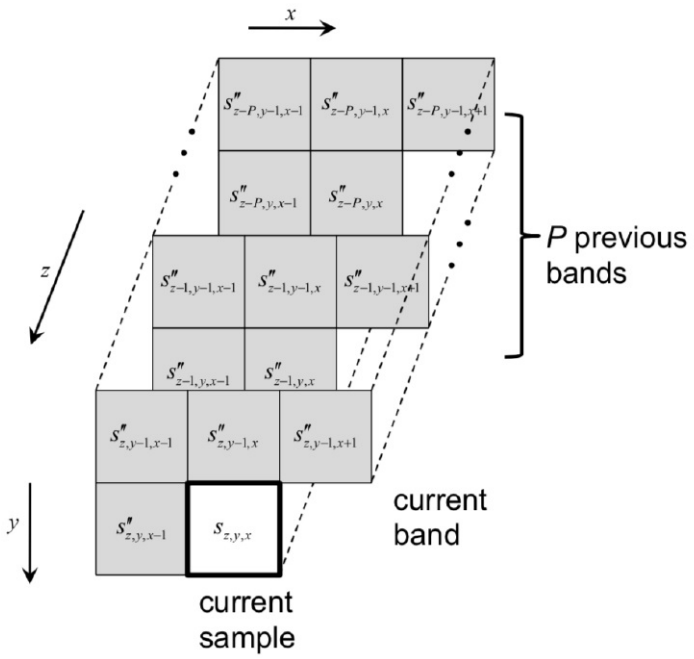

Figure 2.4. Coordinates system used for input samples.

main differences are that wide neighbor-oriented allows to reach the best coding efficiency, narrow neighbor-oriented permits a easier hardware implementation since it avoids data dependencies, while column-oriented reduces the effect of striping noise.

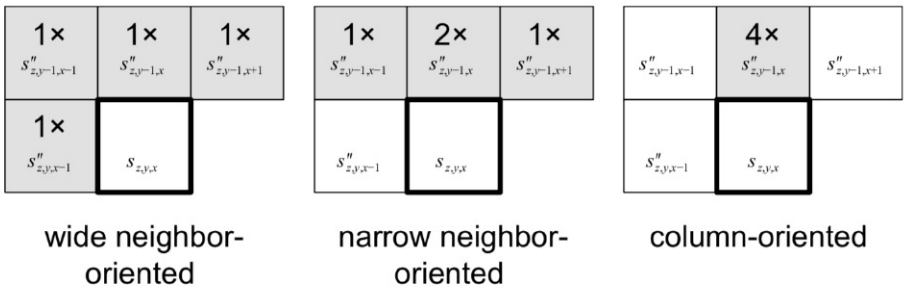

Figure 2.5. Coordinates system used for input samples.

There is another vector defined as local differences, that consists in the difference between a sample representative in the neighborhood of the current pixel and the

average local mean. Differences are computed in the same spectral band (same z value), using the neighbouring pixel on the top (y-1), on the left (x-1) and top left (x-1, y-1), with respect to the current pixel to encode. In order to avoid division, which can be expensive in terms of time delay and hardware implementation, the local difference is computed as 4 times the sample representative, minus the local sum.

There are two ways to use the local differences: full and reduced. In the full mode, all local differences are used (three in the same spectral channel and P in the previous channels), providing better performances for multispectral imagers, whisk-broom imagers and calibrated imagery. In the reduced mode instead, only the P local differences from the previous spectral channels are employed, which it is preferable for raw data from pushbroom hyperspectral imagers.

In a nutshell, the local difference vector contains the spatial/spectral neighbouring samples of the one to be estimated with their (estimated) mean value removed.

The prediction residual can be evaluated as $\Delta_z(t) = S_z(t) - \hat{S}_z(t)$, where $S_z(t)$ is the original pixel, while $\hat{S}_z(t)$ is the predicted sample. $\hat{S}_z(t)$ can be evaluated from the predicted local differences $\hat{S}_z(t) = W_Z^T(t) \cdot U_Z(t)$, where W_Z^T is the weight vector, while $U_Z(t)$ is the local differences vector.

The prediction residual is used to update the weight vector by means of the sign algorithm; the new weight vector will be used for the prediction of the next sample to compress.

2.2.2 Compression code

The compressor C code is divided into multiple files that will be briefly described in this section. Since the fastest way to validate compression performance is to decompress the compressed image and compare it with the original one, also the decompressor is implemented in C code.

Compressor Main: This is the main file in which each function of the compressor is called. This program is intended to perform all the operations needed for a single image that has to be compressed. For instance if there are five images to process, this program must be run five times. In the file, the first part is dedicated to the parse and to the validity check of the input, output, encoder and decoder parameters. If all the parameters are inserted correctly, then the algorithm of compression starts, which is called predict_NO_RC (contained in file predictor.c). Subsequently, dump residuals are written in a file and all the structures are deallocated.

For the inputs, the first parameter to set is the image path, from which the image parses. Input image can be provided in BSQ or BIL order and this parameter has to be specified after all other image parameters are reported, like the spatial dimensions of the image in number of pixels, the number of spectral bands, the number of bits, the endianness and the dynamic range used to represent the pixel

values.

Input pixel value can be provided as unsigned or signed, in the latter they are converted into unsigned by summing an offset equal to half of the dynamic range of the values.

Output parameters are linked to the output image path, in which the image is stored. For this reason the name of the compressed image must be specified.

The predictor must receive some working parameters among them, such as the number of prediction bands to be used, the prediction mode (full/reduced) and the type of local sum (wide/narrow and neighbour oriented/column oriented). Also weights file path and their parameters, such as the resolution and optionally custom initialization values, must be provided.

The encoder algorithm must be set with a parameter that selects among sample adaptive encoding, hybrid encoding or range encoding. Subsequently, also proper working parameters for the chosen algorithm must be passed.

Predictor: The predictor function iterates over the image pixels following a BIL ordering (Band Interleaved by Line). The high level routine, that performs the prediction, is the function predict_NO_RC, in which all sub functions are called step by step. The first function is the local_sum with which the differences for all the prediction bands are evaluated. As abovementioned before, the local sum is computed on representative samples, not with the original values, and it is performed differently in accordance with the mode chosen among narrow/wide and neighbour oriented/column oriented. If the full mode is chosen, local differences are evaluated, otherwise this step is skipped with the reduced mode. Then the function compute_predicted_sample is called, in which the predicted local differences are computed using weights, obtaining predicted samples from them. If the encryption is selected, there are some functions calling the SpongePRG function, that generate random bits used to flip the sign before quantizing the predicted samples. The next step is to compute residuals and map them to unsigned integer, using the function compute_mapped_residual. Finally, mapped residual are passed as input to encode_pixel_sample (contained in sample_adaptive.c) that encoded them using an entropy coder.

Entropy Coding: The used coder is based on the sample adaptive entropy coding algorithm, all coding and initialization functions lies in the sample_adaptive.c file.

Sample Representative Evaluation and weight update: Starting from the mapped residual, samples are passed to the local decoder that reconstructs the original dequantized samples. They are passed to the function compute_sample_representative, which computes the sample representative. These values are useful to evaluate differences for the following samples. After, all weights are updated with the function update_weights, adapting this way the weights to the current sample. Once all the previous functions are called, the algorithm repeats them until all pixels of the image are processed. This was the last step.

Once the image is compressed, the software deallocates all the variables and the data structures, ending the process.

Only the compressor requires a VHDL code implementation on the FPGA, but, in order to debug the correctness of the compressor, also the decompressor should be designed in C code. **Decompressor Main**: Its function is to reconstruct the original image, starting from the compressed file. This code receives less parameters with respect to the compression main, because many parameters are obtained reading the header of the compressed file with the function readHeader.

Decoding Entropy Coding: this is the specular part of the entropy coding. The function decode_sample_adaptive contained in sample_adaptive.c file is used to reverse the entropy coding. These functions decode the whole body and they store all the decoded residuals in an allocated vector called residuals.

Unprediction: This is the reciprocal of the predictor. Starting from the residual, it is possible to reconstruct the original image. Functions contained in unpredict.c repeats the same logical step of the predictor and they iterate functions untill the whole pixels are reconstructed, using a BIL ordering. When the whole image has been reconstructed, the software deallocates all the used variables and terminates.
All these codes are called by bash file test and codec2_NL, in particular the following parameters are used :

- Number of Bands per Prediction $P = 3$;
- Register Size $R(inbits) = 64$;
- Weight Resolution $\Omega = 19$;
- Weight update scaling exponent change interval $tinc = 64$;
- Initial weight update scaling exponent parameters $vmin = -1$;
- Final weight update scaling exponent parameters $vmax = 3$;
- Prediction mode Full Wide/Neighbour Oriented;
- Sample representative parameters all set to 0;
- Use of non-band dependent absolute error limit.

  Sample adaptive Encoder with the following parameters:

  - Unary length limit $Umax = 18$;
  - Initial count exponent $Y_0 = 1$;
  - Accumulator Initialization Constant $K = 3$;
  - Rescaling Counter size $Y^* = 6$.

2.3 Encryption and Keccak algorithm

The main task of the code to implement on the board consists of compressing received raw images and encrypting them in a secure way. The encryption is based on the Keccak library, that refers to a family of sponge functions [10] deriving from a generalization of hash functions, which are fundamental in modern cryptography. Generally, starting from a message of random length encoded as a binary string, a hash function will compute another binary string of fixed length, which is conventionally called a digest. In figure 2.6 there is a graphic representation of hashing procedure.

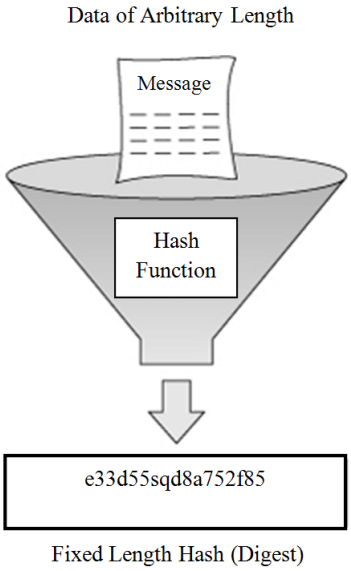

Figure 2.6. Basic concept of hashing function.

Usually hash functions are used in message authentication where the digest is used as a fingerprint for the original message and sent with it to a receiver. The receiver of the message can check its integrity by hashing the received message and comparing the obtained digest with the sent one. If the digests are equal, the message is then identical to the original one, otherwise it has been altered by someone or corrupted during the transmission.

In order to use this method, the hash function must fulfil some important rules, in particular the code should be accessible and understandable by anyone, so that the functions can be distributed and used by both senders and receivers. Furthermore, hash function should be robust enough in such a way that the same digest corresponds to two different messages.

Using the idea of hash function, which has a fixed output length and stream

ciphers with a fixed input length, the sponge function has been created. It performs the same operation with the benefits of variable input and output length management.

2.3.1 The sponge construction

Keccak functions can be used as stream ciphers, hash functions, Message Authentication Codes (MAC) and pseudo-random number generators, depending on the configuration. The main part of the function is based on a fixed length transformation and a padding rule for the input, which are capable of mapping from a variable-length input to a variable length output. In general the function sponge refers to the way in which it elaborates input messages, in fact it is iteratively absorbed into the state with a fixed size throughout multiple rounds of a simple round function before the digest is squeezed out.

The sponge construction [11] is described as a function $sponge[f, pad, r]$ that uses a fixed-length transformation, also called transformation f, a known rule for padding the input that can change with the use of the function and a parameter r, called bitrate. A finite-length output can be obtained by truncating it to its l first bits.

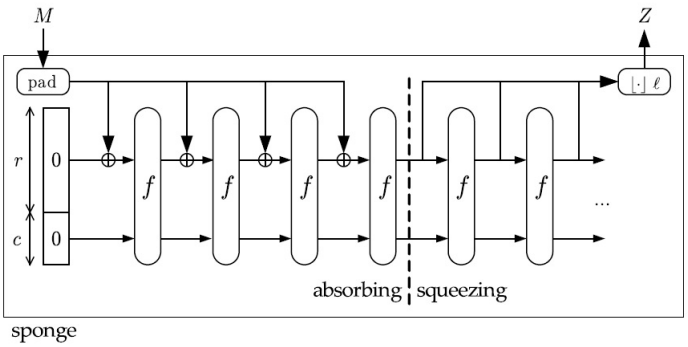

Figure 2.7. Sponge construction diagram.

The permutation f works on a fixed number of bits with width b called state. In figure 2.7 the schematic blocks of the sponge construction are provided. M is referred to the input message, Z is the output message, b is the length of the state and f the permutation function. The state is divided into two blocks of bits that are processed in a different way: the inner state, also called capacity part of c bits, that are never modified and can not be read, and the outer state, known as rate part of r bits, that are permuted by the function f and sent to the output. At the beginning, the state vector is initialized to zero. Then the input message is divided

into r bits blocks and padded. After these initialization procedure, sponge method can be divided in two main phases:

- **Absorbing phase**, in which input message block of r bits is XORed with the rate part of the state and passed to the permutation function, which overwrites this value with the permuted one, while the capacity part remains unmodified in the state. When all blocks of the subdivided input message are processed, sponge construction passes to the squeezing phase;
- **Squeezing phase**, where the rate part is returned to output and interleaved with the applications of the permutation function f. The number of iterations depends on the number of bits l requested at the output.

The capacity part c actually establishes the security level of the sponge construction. Requesting $l < c$ bits at the output it is possible to avoid multiple squeezing phase, in particular if the output length is $l = \frac{c}{2}$ the sponge function reaches the maximum security level.

2.3.2 Keccak family sponge functions

The main advantages of the sponge construction are the high flexibility and security. Any instance of Keccak sponge function family [12] uses one of the seven Keccak permutations defined as $Keccak\text{-}f[b]$. Starting from the same input message and changing the permutation, the digest differs. This is due to the fact that each permutation defines different parameters for the sponge construction, in particular it specifies the size of the state, the partition of rate, the capacity part of the state and the number of rounds of the functions.

These $Keccak\text{-}f$ permutations are iterated constructions based on a sequence of almost identical rounds. The number of rounds n_r depends on the permutation width, given by $n_r = 12+2l$, where $2^l = b/25$. These seven permutations are defined using the b parameter that changes in range $b \in \{25,50,100,200,400,800,1600\}$ and corresponds to the width of the permutation, that is the number of bits used in the state.

The state vector can be considered like a 3D parallelepiped array with coordinates rule, as illustrated in figure 2.8. Each bit of the state is represented by one cube of the 3D array. b parameter is defined as $b = 5x5xw$ where w is the length of the data input, also called lane, and it can be in range $w \in \{1,2,4,8,16,32,64\}$ and is placed along the z axis. This can be easily understood in figure 2.9, where there is the representation of basic blocks.

Due to the definition of b, each plane x-y, called slice, can be considered a square with 5 cubes (bits) and this remains fixed for all Keccak definitions. The only variable that changes is the input word length w, which consequently changes the lane (z axis) dimension. When the input message is read lane by lane, all the

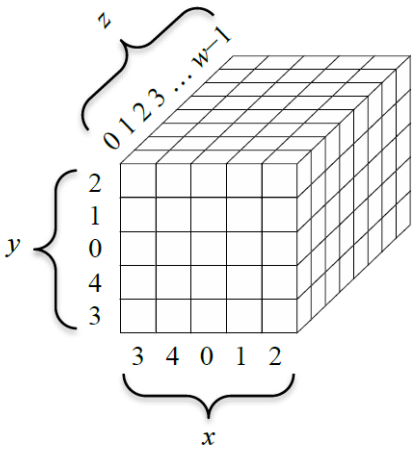

Figure 2.8. State 3D vector of Keccak function.

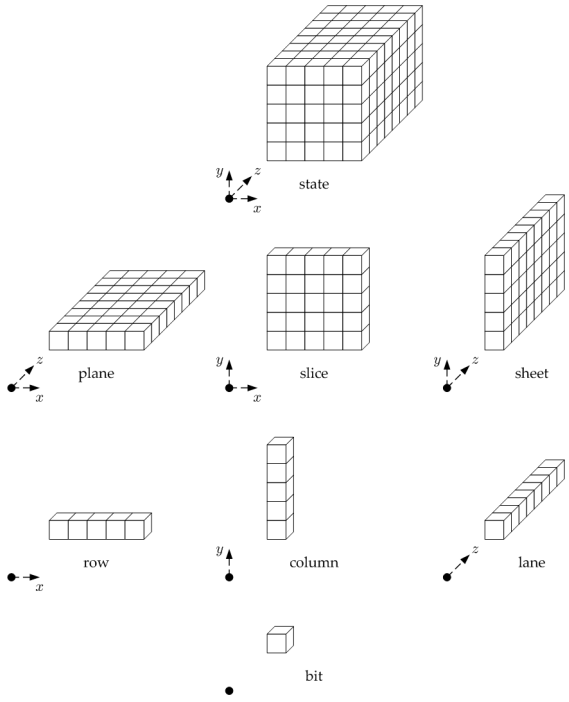

Figure 2.9. Definition of smallest block of state array with an example with w=8. / Keccak-images

state is filled up and Keccak rounds are applied to this structure using 5 equations : Theta,Rho,Pi,Chi,Iota. In particular Theta supplies diffusion by XORing adjacent columns in bits, Rho and Phi provide dispersion, Chi using AND,XOR and NOT logic operator to each bits applies a non-linear mapping, while Iota merely makes

each round different for the others changing the terms in the middle lane and using different constants.

Since the strength of Keccak depends on the length of state, the most safe permutation is the $Keccak-f[1600]$, which is used in the code for the same reason. Moreover it is the safest encryption function adopted in Secure Hash Algorithm (SHA-3) [13] by National Institute of Standards and Technology (NIST) [14] [15]. With $b = 1600$ the number or round required for a secure algorithm is 24 and the length of input words is $w = 64$.

For sake of simplicity the state vector will be described as a 3D vector with coordinates x,y,z: $state[x, y, z]$.

Theta

θ is the first round applied and it takes as input the state vector. This step basically consists on overwriting the state vector by XORing the parities of two adjacent columns with each bits of the third column in the middle as illustrated in figure 2.10. All Theta round can be synthesized in three different steps:

1. Parity of each column is evaluated $P[x, z] = state[x,0, z] \oplus state[x,1, z] \oplus state[x,2, z] \oplus state[x,3, z] \oplus state[x,4, z]$;

2. A second plane $[x, z]$ is generated, where each bit is a XORing result between $[x - 1, z]$ and $[x + 1, z - 1]$ parity bit previously evaluated $plane[x, z] = P[(x - 1), z] \oplus P[(x + 1), (z - 1)]$;

3. Finally original bits of the state are XORed with evaluated bits of the new plane. $Theta[x, y, z] = state[x, y, z] \oplus plane[x, z]$.

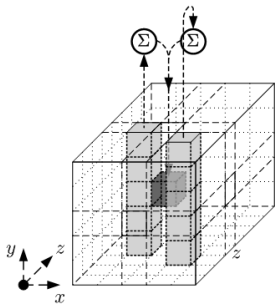

Figure 2.10. Application of Theta round on the state vector with w=8. / Keccak-images

Rho

ρ is the second step, whose effect is to shift the bits of each lane by a length, called offset, that depends on the coordinates x and y of the lane as in table 2.1. Therefore each bit of a lane is shifted along z-axis by adding to z coordinates an offset, extracted from this table using modulo of lane size. After modifying by Theta the inputs of this function, it becomes the state vector and the result is passed to Pi. This can be described as $Rho[x, y, z] = Theta[x, y, (z - table\_constant)\ mod\ w]$.

	X=3	**X=4**	**X=0**	**X=1**	**X=2**
Y=2	25	39	3	10	43
Y=1	55	20	36	44	6
Y=0	28	27	0	1	62
Y=4	56	14	18	2	61
Y=3	21	8	41	45	15

Table 2.1. Rho rotation offset corresponding to X and Y coordinates with w=64.

A display explanation of Rho mapping procedure is in figure 2.11. The black dots indicate the bit on which the offset is added, while the tip of the arrows points the position after adding of offset.

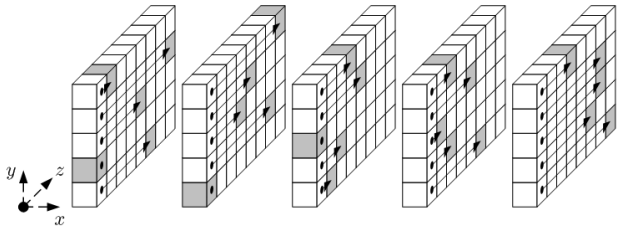

Figure 2.11. Application of Rho round on the state vector with w=8. / Keccak-images

Pi

Third step is function π, in this phase, as described in figure 2.12,the position of lanes of state vector are shuffled and rows are turned in columns. Basically, this function can be seen as $Pi[(2x + 3y)\ mod\ 5, y, z] = Rho[y, x, z]$

Chi

χ takes in input the state vector modified by Pi and applies a non linear transformation. As it can be seen in figure 2.13, this function is based on NOT,AND,XOR

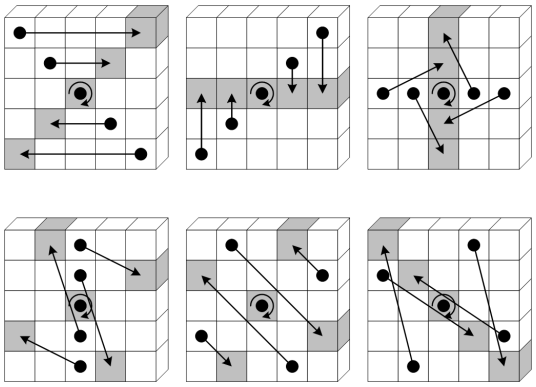

Figure 2.12. Application of Pi round on the state vector with w=8. / Keccak-images

logic operators and it performs XOR operation of each bit with a non linear function of the other two bits in its row. Chi can be written as $Chi[x, y, z] = Pi[x, y, z] \oplus Pi[(x + 1)\ mod\ 5, y, z] \oplus Pi[(x + 2)\ mod\ 5, y, z]$.

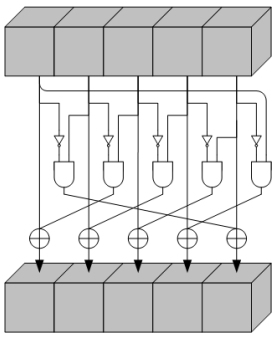

Figure 2.13. Application of Chi round on the state vector with w=8. / Keccak-images

Iota

ι is the last step. This function receives as input the state after being permuted by all other functions. In this step the central lane is XORed with round constant that can be obtained by table 2.2. This final phase is important because in this way diversity of rounds and no symmetry along the z axis are guaranteed. $Iota[0,0, z] = Chi[0,0, z] \oplus round\_constant[z]$

Actually, NIST has approved for the SHA-3, four versions of Keccak sponge function family : SHA3-224,SHA3-256,SHA3-384 and SHA3-512 [16]. The number added to SHA3 specifies the length l of digest requested by user at the output. All these versions use the same permutation Keccak-f[b], where $b = 1600$ is the largest

1 :	X "0000000000000001"	13 :	X "000000008000808B"
2 :	X "0000000000008082"	14 :	X "800000000000008B"
3 :	X "800000000000808A"	15 :	X "8000000000008089"
4 :	X "8000000080008000"	16 :	X "8000000000008003"
5 :	X "000000000000808B"	17 :	X "8000000000008002"
6 :	X "0000000080000001"	18 :	X "8000000000000080"
7 :	X "8000000080008081"	19 :	X "000000000000800A"
8 :	X "8000000080008009"	20 :	X "800000008000000A"
9 :	X "000000000000008A"	21 :	X "8000000080008081"
10 :	X "0000000000000088"	22 :	X "8000000000008080"
11 :	X "0000000080008009"	23 :	X "0000000080000001"
12 :	X "000000008000000A"	24 :	X "8000000080008008"

Table 2.2. Round constant used for Iota function with w=64.

one. What differs between each version is the subdivision in rate and the capacity part of the state. Security depends on the size of the capacity part c, in particular for sure if $l < \frac{c}{2}$ the digest can be tampered.

The relation between the capacity and the rate part is $r = b - r$. For this reason, four versions are characterized by :

- $l = 224 : r = 1152$, $c = 448$, $d = 28$;
- $l = 256 : r = 1088$, $c = 512$, $d = 32$;
- $l = 384 : r = 832$, $c = 768$, $d = 48$;
- $l = 512 : r = 576$, $c = 1024$, $d = 64$.

d parameter is called diversifier and, as the name suggests, it is used to diversify Keccak function and it is in range $0 \leq d \leq 256$. For example two Keccak instances with same value of r and c parameters, but with different value of d, behave as two independent hash functions. If no parameter of Keccak function is defined, default values $r = 1024$, consequently $c = 1600 - 1024 = 576$ and $d = 0$ are used.

2.3.3 Padding rule

Keccak sponge functions apply a multi-rate padding rule to the input message. This rule, called **pad10*1**, is used to adjust input length and also as termination sequence for input message.

If the length of input message is equal to the rate length r, all bits of the message are copied in an inner vector and then XORed with the state. In this case a new

string with length r with all pad bits must be inserted, after the previous input string in order to advise that the input message is terminated.

In the other case if the input message is shorter than rate part, it has to be extend and fitted to the size of the rate part. Essentially the pad rule consists on appending a 1-bit after the last bit of the input message, after this other n 0-bits are appended to the string so that input message length plus 10... reaches $r-1$. At this point a final 1-bit is added to terminate the string. Usual bit ordering convention is little-endian, so the LSB of a byte is at the lower address. The internal Keccak convention on the other hand is big-endian, so the MSB of each byte is located at the lower address, bringing a reordering of each bit. A better explanation can be provided by the figure 2.3.3.

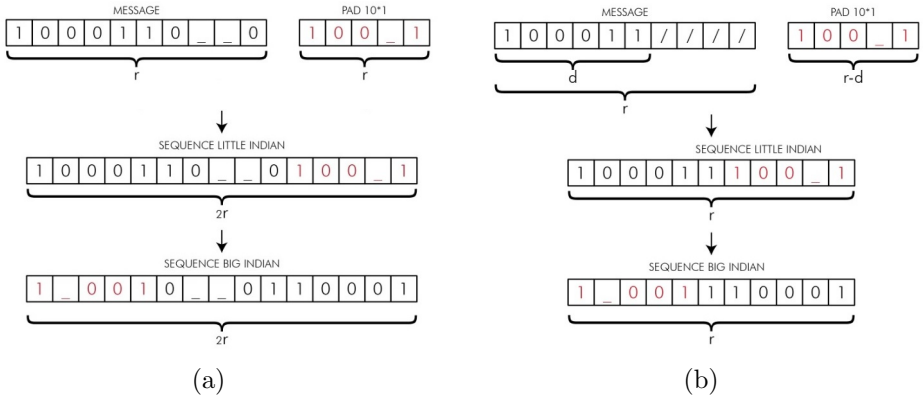

Figure 2.14. a) Padding rule 10*1 applied when the input message is equal to r bits and b) when it is less than r bits, in case of little-endian and big-endian.

2.3.4 Duplex sponge and PRGsponge

In the code the spongePRG function is used as a variant of Keccak sponge family. In particular this function implements a pseudo-random bit generator based on Keccak. This is based on the duplex construction $Duplex[f, pad, r]$ [17] that, like in the sponge construction, uses a fixed-length transformation (or permutation) f, a padding rule "pad" and a parameter bitrate r. Differently from the sponge function, in which there is no state memory between calls, the duplex function has function calls that take an input string and return an output string, depending on all previous inputs.

Each instance call of duplex construction is considered as a duplex object, denoted as D with a state of b bits, σ as input string, Z as output and l the requested number of bits as in figure 2.15.

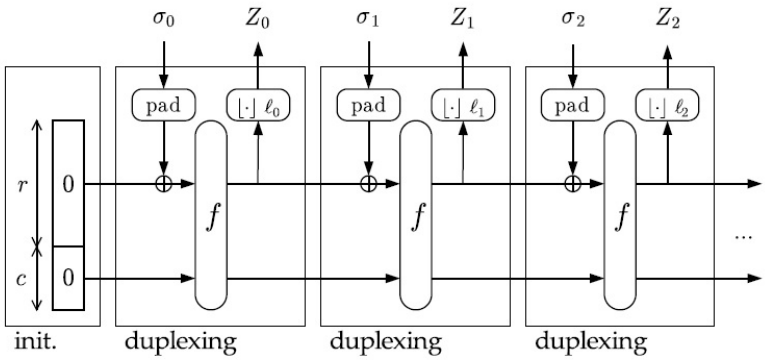

Figure 2.15. The Duplex construction.

The maximum number of bits l that can be requested from a duplex block is r, allowing each block to process a single r-bit input block. For instance, if the input message is subdivided into 5 r-bit blocks, duplex requires 5 D blocks. After initializing the state at zero like for sponge, input message is padded and sent to a D duplex block, where it is XORed with the first r bits of the state. Then f permutation is applies to the state, like in sponge construction, and it returns the asked bits.

It can be demonstrated that the same output of a duplex construction call can be obtained from a sponge call, assuming some conditions on the input message. If the input of the i-th duplex function call is called as (σ_i, l_i) and Z_i is the corresponding output. it can be assert that:

$$Z_i = D(\sigma_i, l_i) = sponge(\sigma_0||pad_0||\sigma_1||pad_1||....||\sigma_i, l_i)$$

This can be proved by considering that, at the begging, both sponge and duplex functions have to initialize the state to zero. After that both functions take as input a block of r-bits with padding rule, which is XORed with the first r bits of the state and finally the f permutation is applied to the state in both constructions. At this point, sponge function and duplex object have the same state and both return the same output, that are the first r bits of the state. Since the sponge function stops at this point while the duplex can proceed with other additional call, it can be considered that the call of $D(\sigma_0,\ l_0)$ is equal to the sponge call that has absorbed a single input $\sigma_0||pad_0$.

Now with the same idea, it is possible to extend the demonstration to a larger message, assuming that after a call of a duplex function $D(\sigma_{i-1},\ l_{i-1})$, its state is equal to the state of sponge function, after absorbing $sponge(\sigma_0||pad_0||\sigma_1||pad_1||....||\sigma_{i-1}, l_{i-1})$.

Once made a new call $D(\sigma_i, l_i)$, the block $\sigma_i||pad_i$ is XORed with the actual r bits of the state and then the f permutation is applied to the current state. This demonstrates what it is asserted in the equation before and it proves that a duplex call $D(\sigma_i, l_i)$ is equal to a sponge call with input $\sigma_0||pad_0||\sigma_1||pad_1||....||\sigma_i, l_i$. Moreover the function spongePRG, that is used in the code, based on the duplex construction, can be built on the sponge function with a little modification, as can be seen in figure 2.16.

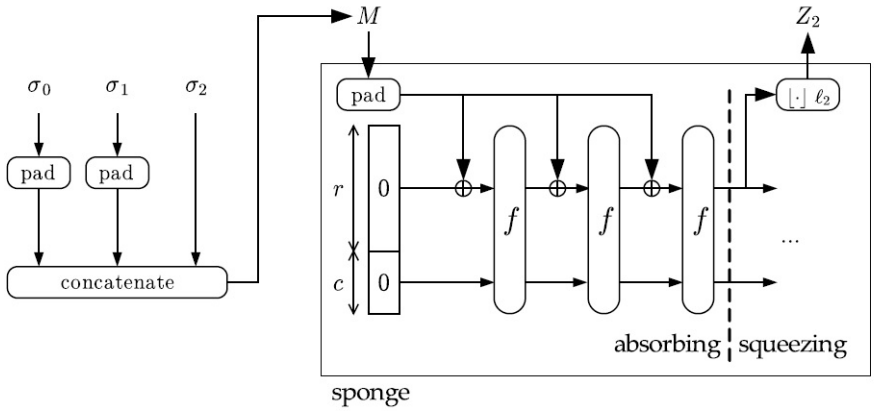

Figure 2.16. Generating the output of a duplexing call with a sponge.

A pseudo-random bit generator (PRG) is an essential block in cryptography. Usually random bits are used to generate keys or unpredictable sequence of numbers. In order to guarantee security, since the use of this block in cryptography, generated bits need to be unpredictable, even if PRG functions are known. Moreover, it is suggested to gather the seed using different source of entropy, in a similar way as in cryptography hash functions. At the beginning PRG is initialized with a seed and this is used as a known starting condition from which the generation of random number begins. The main idea of SpongePRG is to built a PRG on the duplex construction. This can be easily done assuming that the seed is the input message and the random numbers are the r output bits, generated in the state after the Keccak permutation. Internally, the function consists of an input buffer and an output buffer. When a **feed** function is called, seed bits are puts in the input buffer and ,once full of r bits, they are passed to the duplex function call. With **fetch** function, l bits are requested from duplex output function stored in the output buffer. Using duplex function, it is possible to perform multiple feed and fetch request with the limitation that the input seed must be a block of r bits.

All c code and Keccak library information and implementation are provided on GitHub [18] by the Keccak team. Now a brief presentation of main SpongePRG functions [19] used for encryption code will be provided to clarify theory discussion

and to describe how are used.

Since the permutation chosen is the $Keccak\text{-}f[1600]$, this will be the prefix to all SpongePRG functions used.

```
KeccakWidth1600_SpongePRG_Initialize(&instance, KECCAK_CAPACITY);
```

This function is used to initialize an object $SpongePRG[Keccak-f[1600], pad10*1, r, \rho]$. KECCAK_CAPACITY is used to define the capacity parameter and in the code it is defined as $c = KECCAK\_CAPACITY = 512$. After this, all the other parameters are evaluated, in particular rate parameter $r = b - r = 1600 - 512 = 1088$ and the available bits that can be read as output with only one call $\rho = 8 * floor((r-2)/8) = 8 * \lfloor 135.75 \rfloor = 1080$ bits. The pointer points to a Keccak instance that creates the duplex structure.

```
KeccakWidth1600_SpongePRG_Feed(&instance, sharedKey, KEY_LENGTH);
KeccakWidth1600_SpongePRG_Feed(&instance, iv_vector, KEY_LENGTH);
```

This function is used to feed the generator with an input seed. KEY_LENGTH is an unsigned int that defines the size of the key. SharedKey and iv_vector are defined as vectors of char of size $KEY\_LENGTH = 32$, so each of them is an array of 256 bits. As specified above, in order to have a key with different entropy source, two calls to feed function are performed with a different key. SharedKey is a key vector that is known to both the sender and the receiver, which must be kept secret and it is changed after a determinate time to maintain security. This is the first part of seed given to the SpongePRG function. Iv_vector , instead, is a key that changes every time a new file has to be encrypted, generating a random number, which is passed as second key to the SpongePRG function. Since this vector changes every times it is generated by the sender, so it has to be passed unencrypted as first part in the encrypted file to the receiver.

```
KeccakWidth1600_SpongePRG_Fetch(&instance, bytes, buffer_length);
```

This function is used to ask to SpongePRG $buffer_length$ bits and to store them in a vector *bytes*. After the feed with the seed is completed, this function passes the whole bits fed and starts the Keccak computation to return the permuted bits.

```
KeccakWidth1600_SpongePRG_Forget(&instance);
```

This function ensures irreversibility. After all the needed bits were obtained by the fetch function, this function is used to reset the state and to make impossible for un adversary to compute the starting key. Every time a file is completely encrypted, this function must be called before starting the encryption of a new file.

2.4 Hardware equipments

In the project document [4] there is the description of FPGA requirements. The system block that requires more computational effort is the image processing and a preliminary evaluation of the resources used by the implementation of the algorithm is performed on Xilinx Zynq UltraScale+ MPSoC ZCU106 board.

Resource	Xilinx Zynq UltraScale+ MPSoC ZCU106	Algorithm used resource
Lookup Tables (LUTs)	230k	92k (40%)
Digital Signal Processing (DSP)	1728	1112 (65%)
Block RAM (BRAM)	312	276 (88%)
Ultra RAM (URAM)	96	96 (100%)
Flip-flops (FF)	460k	70k (15%)

Table 2.3. Preliminary evaluation of used resources and comparison with available resources on ZCU106 board.

From table 2.3 it is possible to notice how RAM resources (both block and ultra) are widely used. In order to have enough resources and to achieve future implementation improvements, the number of minimum resources required is raised up. For this reason the ZU19EG Xilinx Zynq Ultrascale+ MPSoC (XCZU19EG-FFVB1517-1-E) board has been chosen, due to the amount of its FPGA resources, in particular RAM as previously shown in table 2.4

Resource	Xilinx Zynq UltraScale+ MPSoC ZCU19EG
Lookup Tables (LUTs)	522k
Digital Signal Processing (DSP)	1968
Block RAM (BRAM)	984
Ultra RAM (URAM)	128
Flip-flops (FF)	1045k

Table 2.4. Available resources on ZU19EG board

2.4.1 Zynq Ultrascale+ MPSoC

The Zynq Ultrascale+ MPSoC is a particular general purpose family of Xilinx FPGA, created for advanced specific applications. The board can be divided in two main functional blocks: Processing System (PS) e Programmable Logic (PL).

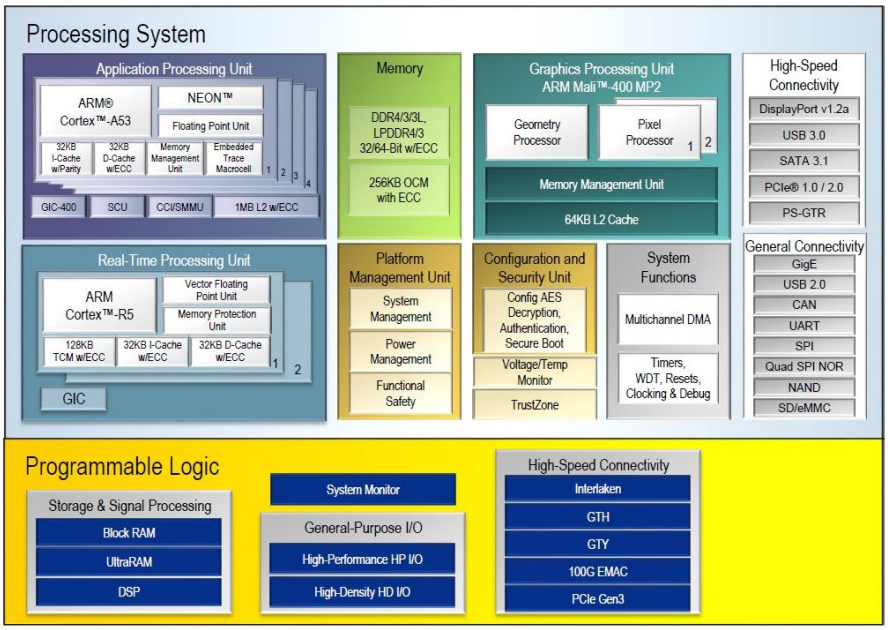

Figure 2.17. Zynq UltraScale+ MPSoCs: EG Block Diagram

Further information can be retrieved from the product guide provided by Xilinx [22], in addition to the FPGA characteristic listed above, and from the figure 2.17.

The PL is the FPGA part. Taking into account the characteristic described in table 2.4, it is dedicated to high performances and to high density general purposes I/O for extension boards. One of this connector can not be used for general purposes, but it is connected directly to a Kingstone-KVR21SE15S8 4Gbytes DDR4 Small Outline Dual In-line Memory Module (SO-DIMM). This type of memory has a smaller outline and thickness than standard DIMM's, a module containing one or several RAM chips on a small circuit board.

The PS is the processor part, which includes a Quad-core ARM Cortex-A53 used as application processor, with a clock operating frequency of 1.5GHz. Since this type of board is multiprocessors, also two other processors are involved: a Dual-core ARM Cortex-R5 used as real-time processor, working up to 600MHz and a Mali-400 MP2 graphics processor. The board provides some peripheral communications, in order to allow debug and communication with the processing unit. In particular there are high speed connectivities like PCIe, SATA, Gigabit Ethernet, USB 3.0 and display port, general connectivity like USB 2.0, SD/SDIO, UART, I2C, SPI and GPIO.

2.4.2 proFPGA motherboard

On the satellite all the boards will be connected using the proFPGA quad motherboard. This kind of board is a complete and modular FPGA system, useful for flexible high speed Application Specific Integrated Circuit (ASIC) prototyping solution. The FPGA are assembled in the dedicated spots and they are plugged to the motherboard, allowing the use of multiple FPGA boards at the same time. FPGA can be all the same or different models, providing high flexibility. Users have fully access to each FPGA, moreover the system offers extensions on top and bottom sides for a specific extension board like DDR-4 memory, PCIe gen1/2/3, Gigabit Ethernet, USB 3.0 or other high performance interface and interconnection boards.

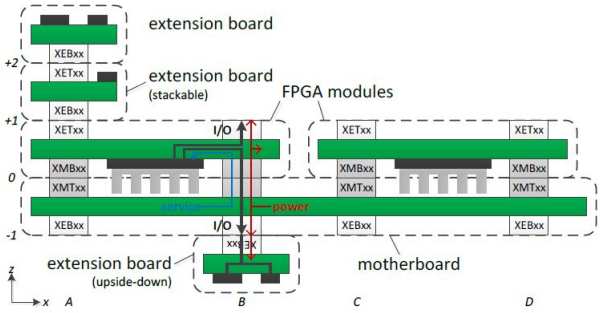

Figure 2.18. Example of proFPGA modular hardware approach system.

From the hardware manual of the proFPGA [21] there is an example of stacked system and in figure 2.18 there is a concept scheme, in particular two FPGA boards and one extension board are connected directly to the motherboard, while two extension boards are stacked one over the other and plugged to the FPGA board. Motherboard offers mechanical fixture, power supply, I^2C-based system management, clocking infrastructure and MMI-64 communication for multiple FPGA modules. Module connectors (grey for motherboard and white for other board in figure 2.18) provide user I/O, power supply and service. User I/O of top and bottom sides are connected to each other, allowing a transparent communication from the FPGA in the top side to the extension board in the bottom side; this way each board can access directly to the extension board without motherboard operations. On the other hand, each FPGA module has access to a Device Message Box Interface (DMBI) communication, called MMI-64, from the motherboard, based on a point to point communication. It is responsible for the System setup, FPGA configuration and for the user communication, offering a high bandwidth and low latency integration of both user software applications and user HDL designs.

Since only one board is used for both compression and encryption, a proFPGA uno motherboard (MB-1M-R2) is used for an easier verification. It has a spot for only one FPGA board.

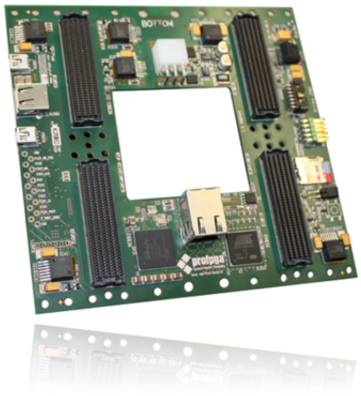

Figure 2.19. ProFPGA uno motherboard.

Due to the presence of a single FPGA board, default settings are used for MMI-64. This type of motherboard provides also :

- 8 extension sites for connecting extension boards (4 on top side and 4 on the bottom one).
- Only one FPGA module.
- 8 fixed clock generators.
- 2 quartz oscillators for different clock references.
- Possibility to synchronize every clock signal with the other one used.
- JTAG port for the Xilinx programmer for FPGA programming.
- USB,UART,Ethernet port for test and communication with the board.
- Automatic boards detections and power protection, very useful for pin assignment and right power settings.

2.4.3 Extension boards

The peripheral communication provided by the board is not enough to connect all the boards of the system together. Moreover some of them have a slow connection, reducing the performances of the entire system. For these reasons some extension boards are plugged into the motherboard. Each board has to be configured and setted in the config file of the proFPGA builder, otherwise the system does not allow power on of the board, and some of them require VHDL block description in order to be connected with the processor. In this section only a brief overview

of the extension board and the connection with motherboard is provided, further details about configuration and implementation will be described in next chapters.

The board coordinate system, used by the proFPGA to identify every extension board connected with extension connector to the motherboard, can be visualized in the datasheet [23].

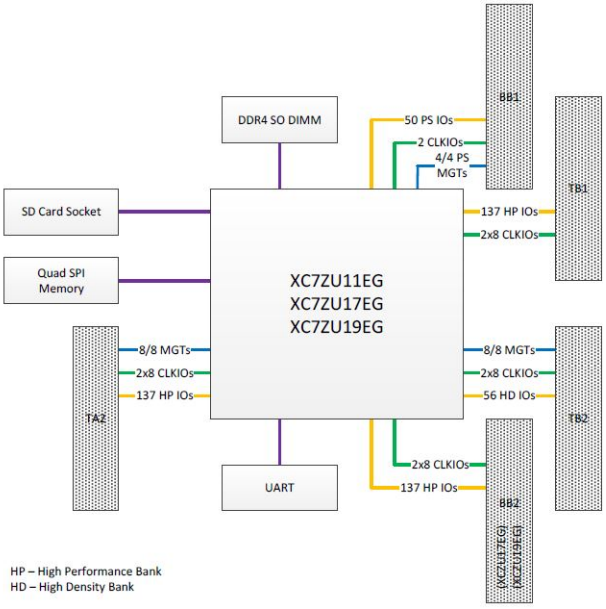

Figure 2.20. Scheme of the coordinate system used on the board to identify extension boards plugged.

Each coordinate consists of 3 letters, that can be schematized in this way : SIDE – X – Y. The SIDE prefix can be 'T' for connection on the 'Top-side' of the board or 'B' for the 'Bottom-side'. The remaining two variables can be considered as the typical X and Y coordinates of a chessboard, the X one is a letter ('A', 'B', ...), while the Y is a number ('1', '2', ...). The name of a module is identified by the code of its coordinate at the top left corner. Since in the project a unoboard motherboard will be used for the testing part, available connectors for both top and bottom sides are: XA1,XA2,XB1,XB2 and the FPGA board is called as *fpga_module_ta1*. Since the top side is used for the SO-DIMM card reader, TA1 connector is not available.

In order to satisfy technical requirements, the following extension boards have been provided for the system:

- GBit Ethernet Board (EB-PDS-GBITETHERNET-R1) (TB2)
- QSFP+ Extension Board (EB-PDS-QSFP+-R1) (TA2)
- Debug Board (EB-PDS-DEBUG-R1) (TB1)

- Zynq US+ Interface Board (EB-FM-XCZUxxEG-R3) (BB1)
- DDR4 Extension Board with 5 Gbyte (EB-PDS-DDR4-R6) (BB2)

The hardware design will be developed by OHB Team. Although, in order to start testing on the board and using it, a preliminary design will be performed and extension boards will be connected as described in the list, following the restriction described into the datasheet [21].

Chapter 3

Tools programs

The thesis is based on different levels of design:

- HW level: it is the lowest level. All boards have to be plugged in the right connector and powered observing datasheet norms;
- VHDL level: this step includes the programming of all FPGA blocks in VHDL. In addition, a VHDL top level with all board pins mapped has to be performed to use HW device on the board;
- SW level: C/C++ code is useful to test some functionalities, to use a peripheral communication device like Ethernet or USB and to perform complex tasks that can not be done in VHDL;
- OS level: an operating system is advantageous for an easy supervision of the board, for simplifying the communication between each board and for providing high level instructions for the device management.

A different software is used for every level, in particular this section provides a short guide for each.

3.1 ProFPGA builder

The proFPGA builder software is used for the HW level and it supplies a graphical environment to create and to run user FPGA designs. In the project the version 2018C is used. It is able to scan the motherboard of the proFPGA and to autodetect all boards connected. After scanning the system, it generates the complete code framework for multi-FPGA HDL designs, including all the scripts for simulation, synthesis and running the design.

In figure 3.1the GUI of the software is shown; in the middle the 3D representation of the system is provided, which is useful to directly identify the boards and

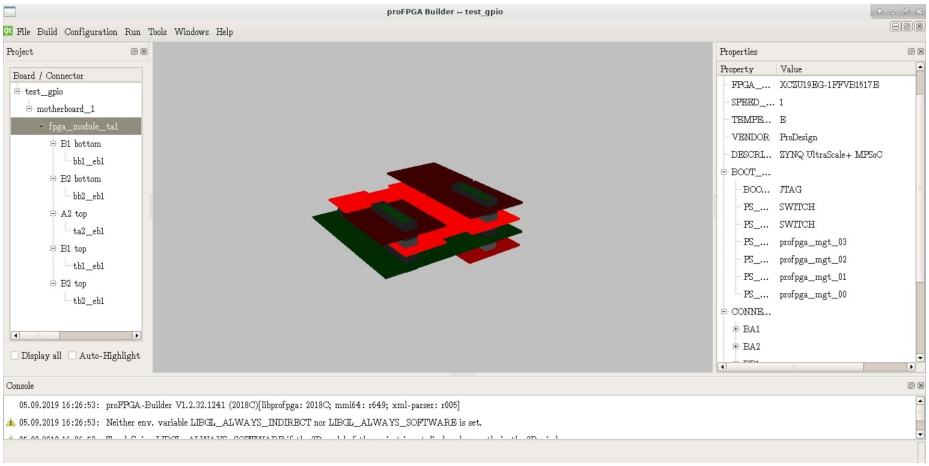

Figure 3.1. ProFPGA builder graphic environment with all the boards connected with their details.

to compare the configuration on the software with the physical system. On the left, the board/connector project window is displayed, with the description of the motherboard and all the connectors used, while on the right the properties windows can be seen, in which, after clicking on the connector, all the details of the board connected to it are displayed.

The first step to perform is the system planning in which HW connections, clocks and power supply are configured. After connecting the motherboard to the PC through the USB cable from port XUSB1 (detailed description on page 55, figure 35 of [21]), selecting the file and then the new project from system, the user should select at this point the TCP/IP connection and click on TCP over USB (in the project the IP address used for the connection is 169.254.0.2). Using this procedure, the Profpga builder creates a hardware configuration file based on the scan of the system connected on the USB port at the IP address defined.

Once analysis of the system is terminated, some settings and a check must be performed in the configuration section. The first thing to define is the FPGA boot mode, which can be loaded in there different modes when it is powered on:

- JTAG, in this way after the power on of the motherboard, the FPGA can be programmed with a Xilinx programmer on the XJTAG1 port (the description can be found on page 55, figure 35 of [21]) in a second time;

- SPI-FLASH, when the motherboard is powered on, it reads from the flash the design for the FPGA;

- SD-CARD, at the power on the system reads from the sd-card the HW description for the FPGA.

Since JTAG mode allows to reload FPGA design without rebooting the motherboard, this method is used for testing the HW design. When an OS is mounted on the board, since it is loaded on the SD card, it is better to use SD-CARD method, so that OS can automatically boot the FPGA.

In this section the reset at the power on and the reset of the system can be also defined. From the ProFPGA instruction, it is better to define them using the physical switch of the board instead of using the software reset.

In the "clock and voltage" section, all the configurations are left as default, so that the software can set and control that boards work properly, in safe conditions and without exceeding constraints.

The last step is to define the plug in settings. In this case the ProFPGA provides two files, that must be used for configuring clock generation SI5338 ,for extension board and for the FPGA.

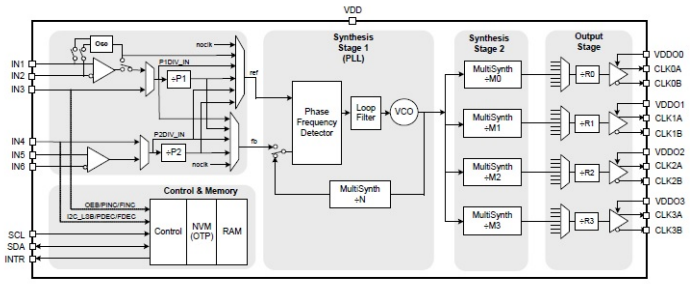

Figure 3.2. Functional block digram of the programmable clock generator SI5338.

From the datasheet, the diagram of the functional block in figure 3.2 can be retrieved. Essentially, this device is a Phase Locked Loop (PLL) with a memory map in which it is possible to define the working function and to obtain the needed frequency. Files provided contain register values that need to be stored in the memory, using an IIC communication, at the power on. Since the system has to work at high performances, 125 MHz frequency, that is the maximum reachable, it is set for communications over Buses between boards connected to the motherboard. For this reason, with the architecture used, SI5338 is programmed in order to provide 125 MHz clock frequency for QSFP and DDR extension boards and 26 MHz, 27MHz and 125 MHz in order to guarantee synchronization with the frequency of the bus.

After this last step, it is possible to run generation file board and to create three different files. The first file generated is the configuration file (.cfg), visible in appendix A, that contains all the setting description for the motherboards, all the extension boards connected with their settings and which connector to use to communicate with the motherboard. At the power on, ProFPGA uses this file to configure and to check the motherboard. If the system connection with extension

boards differs from what is described in this file, the system starting is aborted. This file must be modified every time the system settings change, on the other hand if the system remains unchanged all the settings described above can be omitted and the motherboard can be directly powered on using the configuration file.

The second file generated is the VHDL top level file for the FPGA. This file is useful for the FPGA code implementation. The top level file contains motherboard entity called mb_1_TA1, since there is only one FPGA board with the top right connector defined as TA1. In the port definition of the entity there are all the pins available on the board, each of them having a logical name referred to their own function. By default all pins are defined as in-out, so before using them, each pin must be setted as suggested in the datasheet.

The third file is the XDC constraints file for FPGA, which contains the pin mapping, linking the logical name associated to each pin with its physical definition on the board. All other constraints must be added by hand during VHDL implementation related to datasheet recommendations.

In order to turn on the board, the procedure is to press the run button and then to click on "connect to system..."; in this way the TCP connection is established and the command can be sent to the board. After this, by clicking on start system, the board will be powered on.

3.2 Vivado

In Vivado software it is possible to define the HW block to implement on the FPGA using VHDL language. For this project the 2018.3 version has been used. In order to create a new project, the following steps have to be followed:

- Click on "create new project" and follow the guided steps;
- Select "RTL project" and insert, in add sources, the top level file, while in add constraints insert the xdc file, previously generated with ProFPGA builder;
- Select from the proposed components, in the section part, the used xczu19eg-ffvb1517-1-e FPGA and click on "finish".

Completed these settings, the Vivado GUI appears like in figure 3.3. If the project needs only some HW components implemented on the FPGA without using the processor or extension board, the synthesis and the implementation can be run in order to create the bitstream to load on the FPGA. On the other hand, an HW description of the processor has to be provided and this can be performed by clicking on "create block design" in the section "IP integrator of the project manager".

After that a new blank diagram window appears, in which it is possible to obtain the VHDL code description by drawing and connecting various IP blocks. In particular by clicking the right button, in the blank window a menu appears and

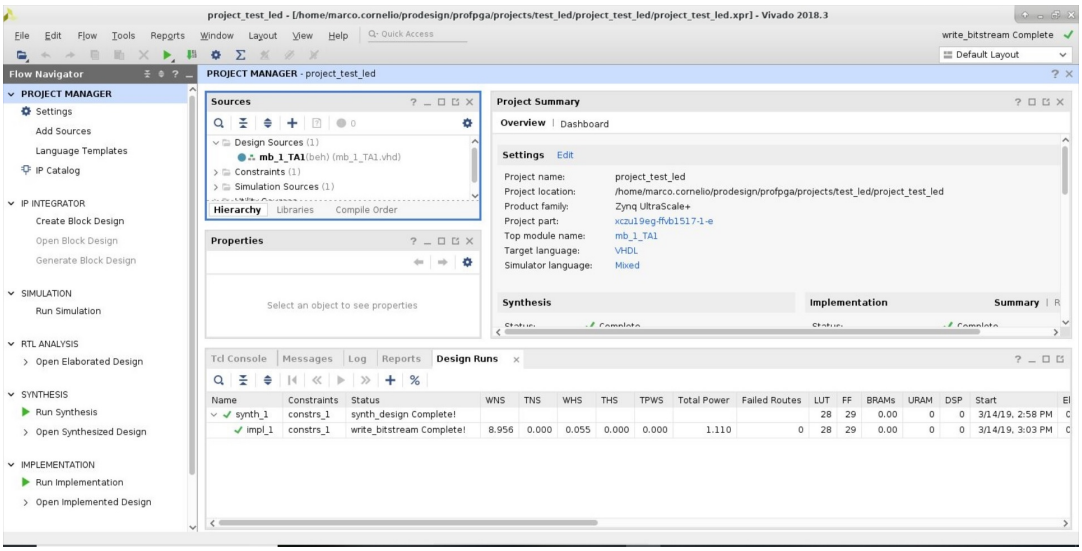

Figure 3.3. Vivado GUI.

by clicking on "Add IP..." and selecting Zynq UltraScale+ MPSoC the processor will be added to the project. Clicking on "Run connection automation" all the pins are connected to make running the board, obtaining a diagram like in figure 3.4, in which the generated digram block file design_1.db. can also be seen.

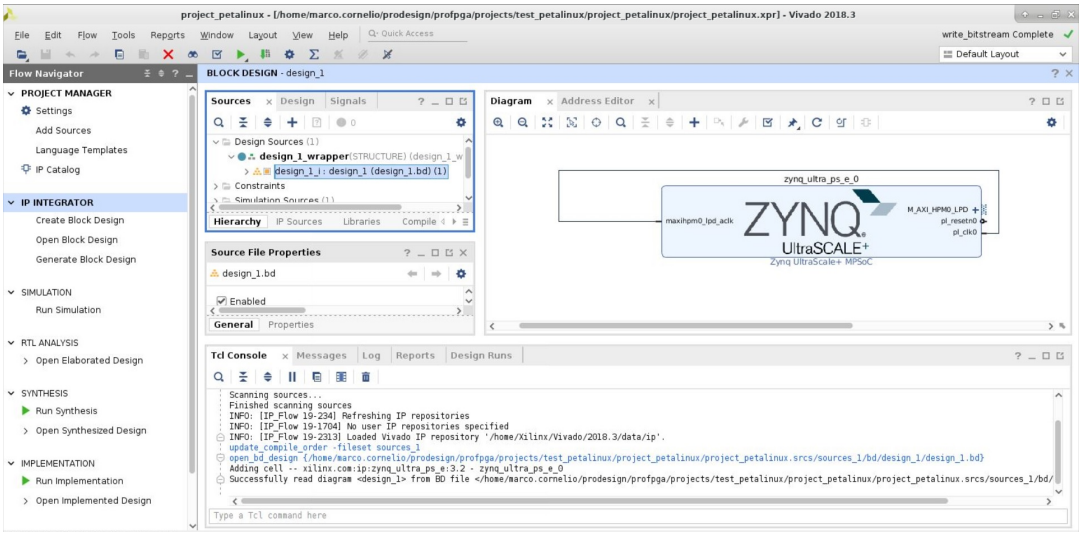

Figure 3.4. Vivado GUI showing block diagram.

Clicking the right button on this file and selecting "create HDL wrapper...", it is possible to obtain the VHDL code of the designed block diagram design_1_wrapper.vhd.

In this file there is simply the creation of the block diagram as a component, which must be declared in the top file VHDL generated by ProFPGA in order to be used.

The next step is to synthesize and to implement the design, performed by clicking on the shortcut that represents a green arrow or by selecting directly the corresponding commands in the flow navigator. If no errors are founded during these steps, a green tick appears on the right in the design runs windows besides the corresponding step.

The last step for the HW design is to generate the FPGA bitstream that contains the FPGA programming information. This can be performed by clicking again from the flow navigator in the "program and debug" session.

At this point the HW prototype is finished and the remaining part to be performed is the board programming, opening the hardware connection and following the guided steps to complete the procedure.

A briefly sum up of the whole flow to follow can be found on Digilent website.

When implementing the IP processor block, many parameters must be set to define on which pins the Ethernet and USB device, that are directly connected to the PS, are connected. Also it is important to define which type of DDR is connected to the board with relatives working parameters. Since it is a custom architecture, there is no information on Xilinx site that explains which parameters to use. From the ProFPGA examples design a command line script is derived, that instantiates and sets all the useful parameters in order to use the FPGA with the Zynq interface board directly connected to PS pins (BB1) and the DDR4 memory via SO DIMM. This file, that can be seen in the appendix B.1, has to be launched from the Vivado tcl command line after that the block digram has been opened by typing "source vivado.tcl".

Before passing to SDK, a folder should be created with all the hardware information, from which SDK can take the implementation details. In order to do this, the essential step is to click on the file, then to export and finally export hardware by also tick to include the generated bitstream. With this procedure in the project folder will be created a new folder with extension .sdk from which Vivado SDK can take system information.

3.2.1 Problems

Using the Vivado 2018.3 version, a little bug of the program was found, which was no reported by any guide or in the Xilinx forum. Launching the program and selecting different boards, it has been discovered that for the most common boards there are no problems, while for the other less mainstream boards the problem is reported in figure 3.5 during the instantiation of the Zynq UltraScale+ MPSoC in the block diagram.

Since the problem appears just instantiating the block without setting any parameters and any other details are provided, the first step was to find where the

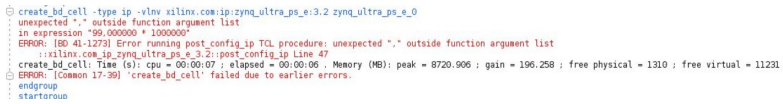

Figure 3.5. Vivado tlc part in which the code and description of the error are reported.

value 99,0000 was set in the block in order to identify the problem. By clicking the right button on the IP, selecting "customize block...", the wanted value is found in the output clock generation, as it can be seen in figure 3.6.

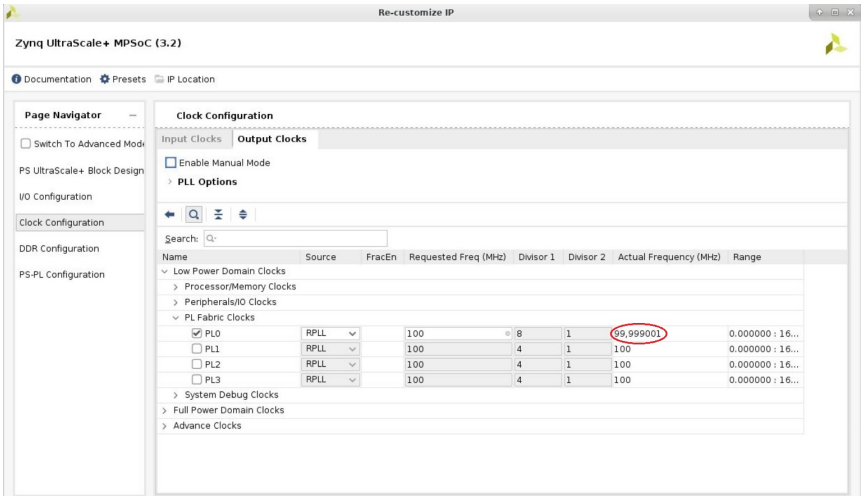

Figure 3.6. Vivado Zynq UltraScale+ MPSoC IP block customization.

The value was impossible to edit since it is obtained by a mathematical division of natural numbers. For this reason, the problem was considered linked to the operating system mathematical calculation. Indeed the problem is solved by changing the language (and also mathematical conventions) operating system from Italian to English (US). Since in the English convention separator between units and decimals is the dot instead of the comma, the IP block can be instantiated with no problems.

3.3 Vivado SDK

By clicking on file and than launching SDK, it is possible from Vivado to enter in the SDK environment. With this tool it is possible to generate code that can be run on the board, to retrieve HW information from files generated in Vivado and to communicate from processor (PS) to the FPGA (PL) and vice versa by using the hardware address. There are two main possibilities to create the software:

standalone and Linux, clinking on "file", then "new" and "application project", a window like in figure will be opened.

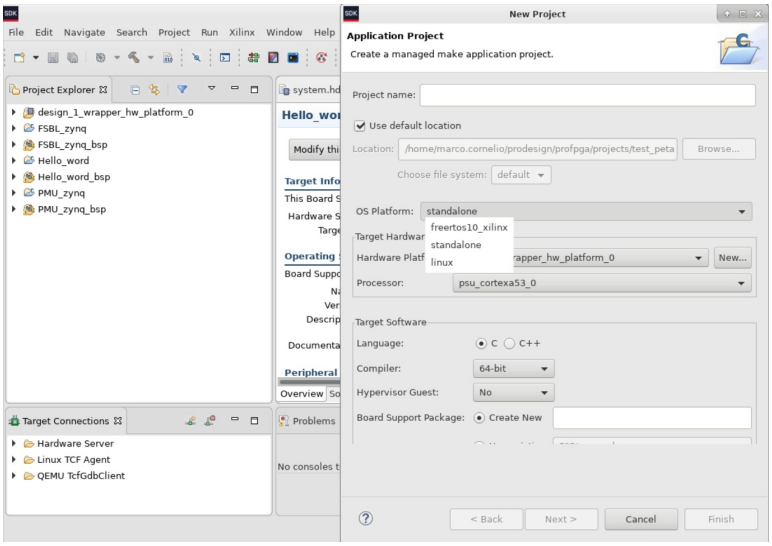

Figure 3.7. Vivado Zynq UltraScale+ MPSoC IP block customization.

The standalone application is used to run the C code over the processor in baremetal mode without any OS running on the board. In this mode it is possible to use the Xilinx library which contains all standard functions to communicate with the hardware of the board. If no OS is used, the program can be loaded directly on the board using the button "program the FPGA" and a communication with it can be established, using the USB connection XUSB1 or XUSB2, that are directly connected to the PS as asserted in [21] at page 113. There is the restriction that both port can not be used at same time.

Communication can be established by using the command screen on Linux:

```
screen dev/ttyUSB0  115200
```

Where ttyUSB0 is the name of the USB device, while 115200 is the value for the baud rate used for the connection, which usually is the higher one. In order to close the connection, the best choice is to press CTRL-A and then K, so in this way the connection is terminated closing the stream channel. Otherwise to start a new communication, it is necessary to unplug the cable and to reconnect it.

The Linux application on the other hand does not allows to use Xilinx library, but it is possible to use all Linux functions. Compiling the code, an executable file with extension .elf is generated, which can be directly loaded on the SD card of the OS boot and run. This is the easiest and fastest method to compile a program for the board, since the compiler for the board is not common, in particular the tool uses ARM V8 compiler and assembler with the command aarch64-linux-gnu-gcc.

Also in this case the communication with the board can be done through USB connector and screen command.

In the project explorer there is a folder name_hw_platform that contains all hardware specifications, in particular there are the files psu_init.c and psu_init.tcl, that are very important as explained in the following section. In the same folder there is also the file named system.hdf, in which there are all the names and the addresses of the hardware design that can be used.

3.3.1 Problems

Some times if the run of the program is launched a second time, this type of error can appears :

Timeout Reached. Mask poll failed at ADDRESS: 0XFD4023E4 MASK: 0x00000010

There is not a clear explanation about the reason why no guide or no Xilinx forum post exist. One solution can be found in a forum regarding the Zedboard [24] in which, without specifying the origin of the problem, the following steps are suggested:

1. Shut down and Reprogram the board;
2. In the psu_init.c file inside name_hw_platform project, comment out the following line as shown and save the file;

```
//mask_poll(SERDES_L0_PLL_STATUS_READ_1_OFFSET,0x00000010U);
```

3. In the psu_init.tcl file, in the same directory comment out the following line as shown and save the file;

```
# mask_poll 0XFD4023E4 0x00000010
```

4. Launch again the program with the run command.

3.4 PetaLinux

After that design is described in Vivado and all codes needed are built on SDK, the last step to be performed is to create an OS for the board, based on the previous characterization. The version of PetaLinux used is the 2018.3, since it is suggested to use one equal to the Vivado to reduce incompatibility error. PetaLinux offers wide flexibility in the OS construction for Xilinx board like Ultra Scale, for this reason there is not a complete flow to follow, but several guides are considered in order to understand which is the best sequence of commands to create an OS that

fits the project requirements and that is not too heavy for the board [25, 26, 27, 28, 29]. An ultimate more detailed reference is published after the test performed on the board [30]. This tool does not have a GUI, but it is all based on command line instructions.

The first step is to change by hand the ProFPGA configuration file and to modify the boot source from "JTAG" to "SD-CARD" in the system configuration, so that the FPGA loads directly boot information from the SD card.

The second step is to decide if First Stage Boot Loader (FSBL) and Platform Management Unit Firmware (PMUFW) should be created in Vivado SDK or let petaLinux to create them. In first case the creation is under the control of the user that can verify the correctness of the procedure, while in the second case petaLinux creates automatically the files with some possible improvements.

The FSBL is responsible for loading the bitstream and configuring the FPGA architecture Processing System (PS) at boot time, without which the boot can not be done.

The PMU, as the name suggests, manages the whole platform, so it has a huge impact on a lot of use cases.

The third step is to launch petaLinux tool and to start the generation of OS using the following commands in the reported order.

```
//project creation
petalinux-create -t project --name name_project --template zynqMP
//move into create project folder
cd name-project
//load hw description
petalinux-config --get-hw-description=sdk_path_folder
//create custom app to run own program on OS
petalinux-create -t apps --name name_program --enable
//substitute in the app folder
    project_spec/meta-user/recipes-apps/name_program/files created example file
    with the program  c file
//setting kernel configuration
petalinux-config -c kernel
//setting rootfs configuration
petalinux-config -c rootfs
//setting device tree configuration (optional)
petalinux-config -c device-tree
//setting u-boot configuration (optional)
petalinux-config -c kernel
//build image
petalinux-build
//creation file boot package
petalinux-package --boot --format BIN --fsbl images/linux/zynqmp_fsbl.elf
    --u-boot images/linux/u-boot.elf --pmufw images/linux/pmufw.elf --fpga
    images/linux/*.bit --force
```

After creating the project, it is important to move to the folder created or to define it as folder path, otherwise petaLinux can not proceed with the operations. After this, the path from which the tool can load all hardware information must be defined, in particular it is preferable to use SDK folder path created by vivado SDK.

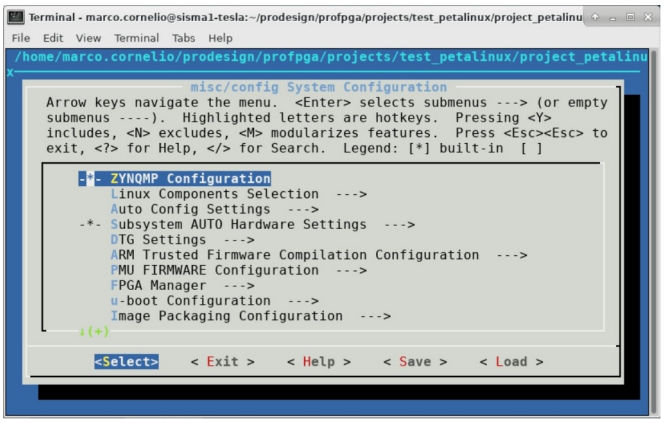

Figure 3.8. Command line showing settings after petalinux–get-hw-description.

In the figure 3.8 the image of the graphical menu is shown, which comes from the petalinux–get-hw-description call. In the project all settings are left to the default ones.

The following step regards the custom instantiation of the user programs. There are two main possibilities for running code on the board:

- Load the compiled program (.elf) in the SD card, run the board and it can be found and launched from the SD card path of the OS /run/media/mmcblk0p1/...
- Using petaLinux command, create the app to define a system call equal to the name app in the OS. This command creates only the template, so the file program must be inserted in the project/spec/meta-user/recipes-apps/name_program/files before launching the petalinux-build. By following this procedure, the program can be called from every point like a system function. Indeed, going in the /usr/bin section of the OS among all available commands, also the user application can be found.

After this, several configuration calls must be performed. The first one is the kernel configuration, followed by the function call, the window as in figure 3.9 will appear in the terminal.

Since the system will use a PCIe connection, the only setting to modify is to enable the PCI, in particular this must be performed:

- Bus support → PCI Support;
- Bus support → PCI Support Express Port Bus Support;
- Bus support → PCI Endpoint → PCI Endpoint Support.

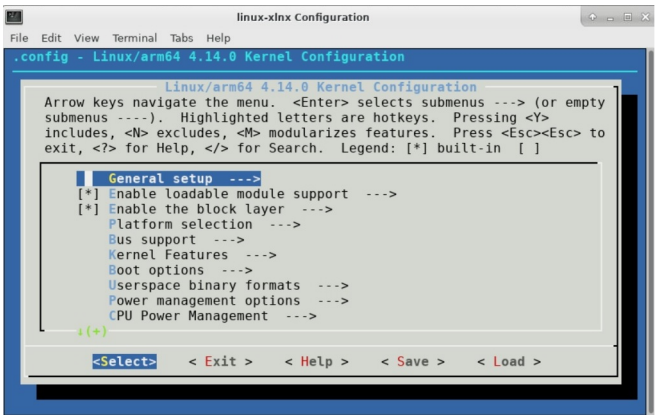

Figure 3.9. Command line showing settings after petalinux–get-hw-description.

The next one is the rootfs config, in which it is possible to add all needed libraries and to custom programs. In figure 3.10 the menu from command line is provided.

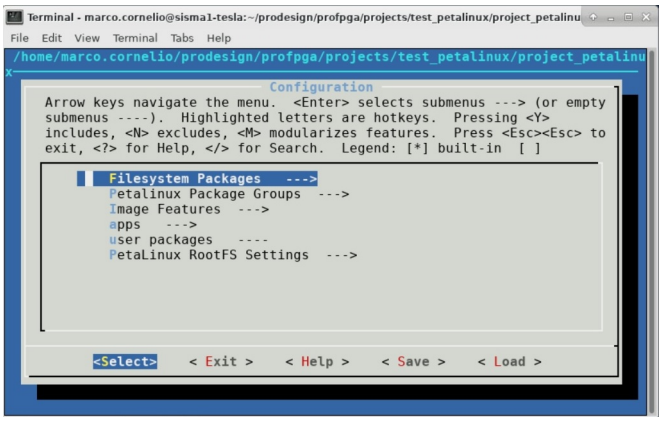

Figure 3.10. Command line showing settings after petalinux–get-hw-description.

Since there are no tips about what to add to the image, after multiple tests the best version for performing test is to add in Filesystem Packages:

- Admin → sudo;
- Admin → sudo-dev;
- Base → shell → bash;
- Console → network → ethtool;

- Console → network → wget;
- Console → utils → man;
- Console → utils → pkgconfig;
- Console → utils → screen;
- Console → utils → vim;
- Misc → gcc-runtime → libstdc++;
- Misc → packagegroup-self-hosted;
- Misc → packagegroup-core-buildessential.

These libraries are added in order to make the test and the debug easier, which does not imply their effective utility in the satellite version implementation. Instead, the ethtool and libstdc++ libraries are fundamental and they must be always included. The former is useful for making Ethernet port working, while the latter for running the C++ code. In order to load user apps on the boot image in the same menu inside apps submenu, the relative apps must be selected.

At this point, the configuration of device tree and the u-boot can be done only in case of changes, otherwise they can be skipped, launching the petalinux-build that proceeds with the analysis and creation of the compiled file based on all the previous information provided. This is a long task that usually takes one hour to be completed.

The final step implies putting together the generated files to load on the SD card by using the petalinux-package command, which takes all the files and creates the image.

Load BOOT.BIN and image.ub on the SD card to insert in the XUSB1 SD-Card Holder ([21] on page 113) placed diagonally over the FPGA. There are two card holders: one is located on the FPGA with a boot functionality, while the other is located on the motherboard which contains its own instructions. The latter must non be removed or overwritten, otherwise this will cause the stop working condition of the motherboard.

Looking in the proFPGA project example material, a bash can be found with the aim of automating the image creation with petalinux. This bash was modified in order to fit the systems requirements. As can be seen in appendix B.2, it is quite similar to the flow described above, with an additional part to substitute in automatic C code for the apps implementation and with a device tree modification in order to make the Ethernet and USB connector visible to the OS.

In order to switch on the board, the first step is to power on the motherboard using the command

```
profpga_run name_config.cfg -u
```

This way the configuration file is automatically passed to the proFPGA, that starts the procedure of powering on. If the board remains off, the power could be forced to on as described in the proFPGA section, even if it is considered a quite brute force approach. Once the boot is completed, the OS asks for login credentials that by default are admin: root and password: root.

Chapter 4

Hardware structure

Before starting with the code implementation, the system must be set up. In order to do so, some test projects are designed, making it possible to understand how the board works and which components are involved to build the required architecture. In this section all the tests performed will be described, starting from the most basic until reaching the most complex one. The starting tests are performed in order to first test the FPGA and processor(PS) separately and then together.

4.1 Base tests : FPGA (PL), processor (PS) and together (PS + PL)

In these designs only the proFPGA and Vivado are used, since at the beginning the aim is to understand how to configure the board without OS. In addition for the FPGA, the debug is performed using LEDS.

Only FPGA (PL) part

In the first test (test_led) a counter was implemented in Vivado, which turns on and off the LEDS placed on it by using the clock of the FPGA. Only VHDL code is used to describe the hardware, without any other blocks. In order to use the clock of the FPGA, some changes have been made to the constraints XDC file and to the top level VHDL.

Since the board uses a differential clock, this can be converted into a single ended, using the IBUFGDS components and a 100Ω termination resistor, that must be specified in the constraint file. Moreover in order to use LEDS, they must be classified as LVCMOS18 elements so that they can be powered with 1.8V. All these changes are described in the [21] at page 30 and at page 122, while all other unused pins are commented.

In top file, as for constraints one, not all used pins are commented. In addition the buffer, the counter and connection with LEDS pins are instantiated in the architecture.

After this step the synthesis is launched and the implementation runs, finally loading the bitestream on the board for tests. All the made codes can be visualized in the appendix C.1.

Only processor (PS) part

In this design the Zynq Ultra scale IP block is implemented, in the Vivado block digram, allowing to use the processor and to write the C code to run over it. No other changes are applied to the constraints file, while the wrapper of the block diagram is added in the top VHDL file with the zynq component.

After this, an example of C code, in which simply "Hello world" is printed on the screen, is implemented in the standalone version of Vivado SDK. Since the Vivado terminal does not work, the only way to see the print is to use an USB connection and a screen command. This code is not provided in the appendix, since only example codes are used.

Processor (PS) and FPGA (PL)

In this paragraph two designs are implemented, that use both PS and PL at the same time. The purpose of both designs is to make a project in which an hardware block is implemented on FPGA to control LEDS, which is programmed with C code running on the processor. In the constraints file only the constraints for the use of LEDS are added, since in these designs the clock of the processor is used instead of the FPGA one. In the first project (test_hello) in the block diagram, a Zynq and an AXI GPIO are instantiated and after the run block automation also the axi interconnect is added to the block design, in order to connect the gpio to the processor. By clicking on the right button and selecting "create port", a vector of 8 bits is instantiated to create a port of 8 bits that goes out of the block created. After that, the wrapper of the block is generated, the implemented component is added to the top VHDL and the output port of the component is directly connected to the pins of the corresponding LEDS of the board. In Vivado SDK a standalone C code project is created and the Xilinx functions are used.

The first step is to define an address variable to communicate with the AXI GPIO component. After that, the device is initialised and its physical address, obtained by the system.hdf file, is linked to the address variable led_blinker. The last step of the initialization is to set the communication direction of the tristate buffer of channel, using the Xilinx function, in which the fist parameter is the address of device, the second is the channel used, while the last one is to define the tristate mask where 0 stands for the output and 1 for the input.

After that the device has been initialized, with the DiscreteWrite function it is possible to write the value for the LEDS, specifying the address of the device, the channel used and the value to be written in the device. All the codes and the block

diagram used can be seen in appendix C.2.

The second design (test_up_down) does the same things, but in this case a custom AXI slave is used to control the LEDS instead of an AXI GPIO. In order to crate an AXI slave block, click on "tools" and then create and package a new IP. In the window that will be displayed, select "create AXI4 peripheral" and select a datawidth of 32 bits and 4 registers, since they are minimum values that can be chosen respect to the real need of just 8 bits for this design. After this, the slave is created and it can be added to the digram block by simply clicking on the right button and selecting in the IP catalog the created block, that in this case it is called blinker.

The created block has to be modified, since a 8 bits output for driving LEDS is needed. By clinking on the right button on the created IP and selecting "edit" in IP packager, it is possible to modify the HW block. The IP is opened in a new Vivado project and in the blinker_v1_0_S00_AXI, that defines the architecture of the AXI device, an output vector OUTPUT_LED of 8 bits is added in the port declaration and, at the end of the file where it is possible to add user logic, this port is connected to the first 8 bits of the first register implemented. In this way when something is written in the register device, this will be forwarded to the OUTPUT_LED vector. At this point only the logical behaviour of the device is modified.

To modify the IP block in the block diagram, also the blinker_v1_0 has to be modified, in particular the OUTPUT_LED logic vector is connected to the instantiated port output_led of the IP block. To finalised these changes, the IP must be repacked, otherwise no modifications are brought to the block diagram even if VHDL files are modified. In order to validate changes, click on "package IP" and then go to one of the sections in which the tick is absent and click on "merge changes from customization parameters wizard". After this, if no errors appear, all the section will be validated by a tick, except for the "review and package" section. So go to this section and click on "repackage IP". If these repackage steps are not performed, the block has not been working as expected.

After this step, return to the block diagram, delete the old version of the block and add the new version to the block diagram.

As in the other design, also here a port has been created for managing the output bits and the component has been added to the top VHDL after the wrapper generation.

Constraints and top VHDL files are the same of the previous design. In the Vivado SDK a standalone C code is created, in which no Xilinx function are used. In this case from system.hdf the address of the blinker block is taken and a pointer variable to this memory location is created. Writing to the pointer, it is possible to change the status of LEDS. Also for this project it is possible to view the block diagram and the files generated in the appendix C.3.

4.2 Test of the interrupt functions

The next test performed is to realize a design, in which the LEDS are changed by an interrupt instead of a running program, that can stock the processor to perform others works. In order to test also the debug extension board, also LEDS on this board are used. In the constraints file all the constraints of the LEDS for the debug board are added as it has already been done for the LEDS on the board. In the block diagram the AXI timer was instantiated, which is useful for managing the interrupt. In order to make the interrupt visible to the processor, it must be enabled in the Zynq IP block by going to the "PS-PL configuration", to "general", to "interrupts" and finally to "PL to PS". By clicking on the "run automation connection", all the blocks are connected in a proper way. For the C code in Vivado SDK, the code example on the Xilinx forum [31] is followed.

In the C code the same steps of previous test are performed for the AXI gpio. For the interrupt management the first step is to define a pointer to the timer device. In the timerInterruptHandler all the functions that must be performed by the interrupt call are defined, which are the functions for the gpio to change LEDS status.

The initialization of the interrupt is made by several parts, firstly the timer is initialized linking the address to the physical address of the AXI timer in the block diagram and then the pointer is also linked to the handler of the interrupt. This way every time the timer reaches the limit, it calls the corresponding handler to be performed. After this, the interrupt is enabled and both the auto reload option and the limit of the counter are setted. Finally the timer starts counting. All the files used can be found in appendix C.4.

4.3 GBit Ethernet Board (EB-PDS-GBITETHERNET-R1)

At this point also the OS is introduced in the design and, after realizing how to built the image and to boot the system from the SD card, the first test project has the purpose of using the GbitEthernet (test_client). The ethernet port is essential in the project for testing the communication between boards, moreover it can be only used with a running OS.

Before starting with the design description, a brief summary of the GbitEthernet extension board must be provided. This extension board provides two Ethernet connectors supporting 10m/100M/1G and MII/GMII/RGMII. Each connector has five status LEDs to advise about PHYsical layer (PHY) status.

For the project, Reduced Gigabit Media-Independent Interface (RGMII) interface physical connection is used between the Ethernet PHY and the Ethernet Media

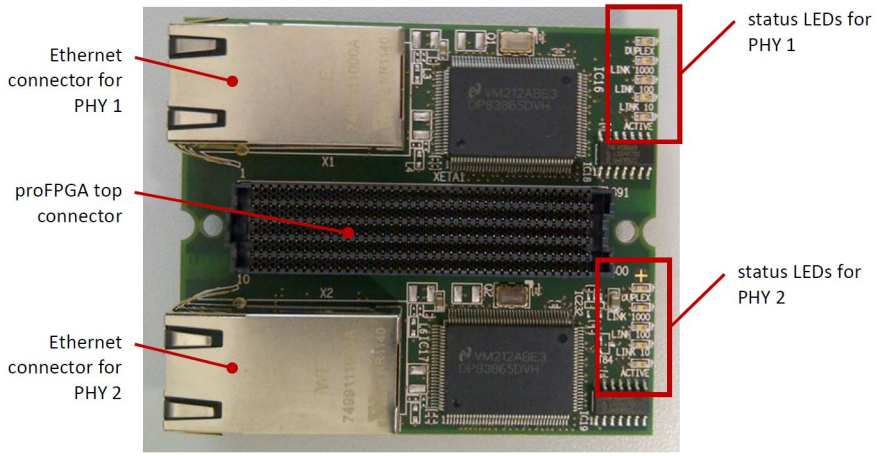

Figure 4.1. Gbit Ethernet board.

Access Control (MAC): in particular the PHY is the DP83865DVH Texas Instruments and the Ethernet MAC is inside the FPGA. The purpose of this interface is to transmit the path, from FPGA to PHY, and to receive the path, from PHY to FPGA, both of them having an independent clock, 4 data signals and a control signal. By clocking data on both the rising and the falling edges of the clock, like in a dual data rate (DDR) memory, and by eliminating non-essential signals (carrier-sense and collision-indication), it is possible to move from 24 pins in GMII interface to 12 in the RGMII, reducing the number of the pins involved by one-half.

The RGMII standard specifies that data and the clock must be given as output simultaneously (ie. without any skew on the clock). Since this condition is not easy to obtain, for a proper sampling of the data signals at the receiver side, the RGMII standard specifies that the skew must be added to the clock signal, either by the PCB traces, or by the receiver itself as can be seen in figure 4.2.

The skew of the TX and the RX clocks can be managed independently, without being implemented at the same stage on each path, but somewhere on each path. For this reason it is critical to understand each of the delay stages in the target system in order to ensure that the clocks in the RGMII interface are properly skewed.

The extension board is connected to the motherboard throw the external connectors, but they are not directly connected to the processor. In the proFPGA architecture there are two different pins, the MIO and EMIO. MIO pins are predefined and it is possible to select, from predefined sets of possible pin connections, the particular PS peripheral which will be directly connected to the PS part. On the other hand there are the EMIO pins that are not directly connected to the processor, but, passing through the FPGA, they can reach the PS part. In this case pins have to be defined where the constraint (XDC) file is located and they must

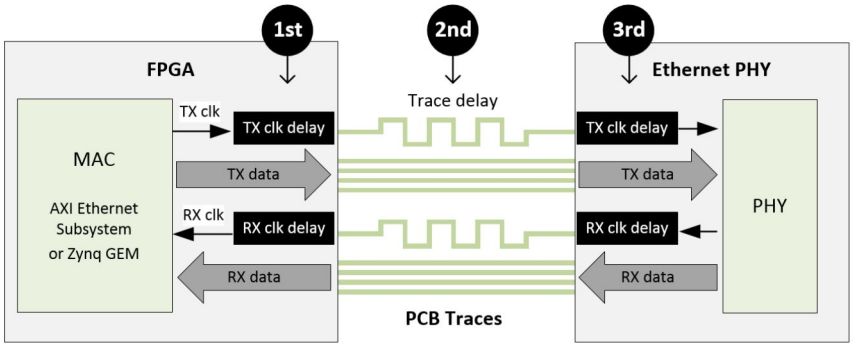

Figure 4.2. Diagram of interface communication between FPGA and Ethernet port with the three possible positions to add skew: 1)FPGA, 2)PCB and 3)PHY.

be connected in top VHDL module with Xilinx tools generated system (PS module) wrapper module. EMIO pins are very useful when the number of peripherals provided by the board are not enough, i.e in our case there are no Gbit Ethernet ports on the board, but only a Gbit interface. Adding the relative IP block and coding it in the PL, it is possible to plug the Gbit Ethernet extension board and to extend the number of ports. Eventually one substantial difference between using MIO or EMIO pins is that, using the last ones for PS peripheral, PL must be configured (FPGA design running) to have pins connected to the PS peripheral (they are wires through PL logic). E.g. Linux device driver (e.g. for UART you have configured to use EMIO) can use that PS UART device only if PL is configured (wire connection through PL logic must exist). On the other hand, MIO pin PS peripheral connections does not require PL to be configured, pins are connected outside PL via PS configuration registers, set up by FSBL.

In order to connect the PS part with the Gbit port, the "Gmii to Rgmii" IP block, in figure 4.3, has to be added to the block diagram.

With this block it is possible to solve an important issue.

- Firstly it is possible to connect Emio pins of the Zynq Ultrascale+ MPSoc block with the pins of the Gbit Ethernet in the block diagram and consecutively in the VHDL top level.
- Secondly it is possible to move from Gmii to Rgmii interface, using less pins as described above.
- Eventually it is possible to add a skew on the TX clock, by accessing in the Vivado GUI through the option “Skew added by PHY”. When this option is ticked, the core will output the TX clock without the skew. When unticked, the core will output the TX clock with 2ns of skew. For the RX clock no options are available.

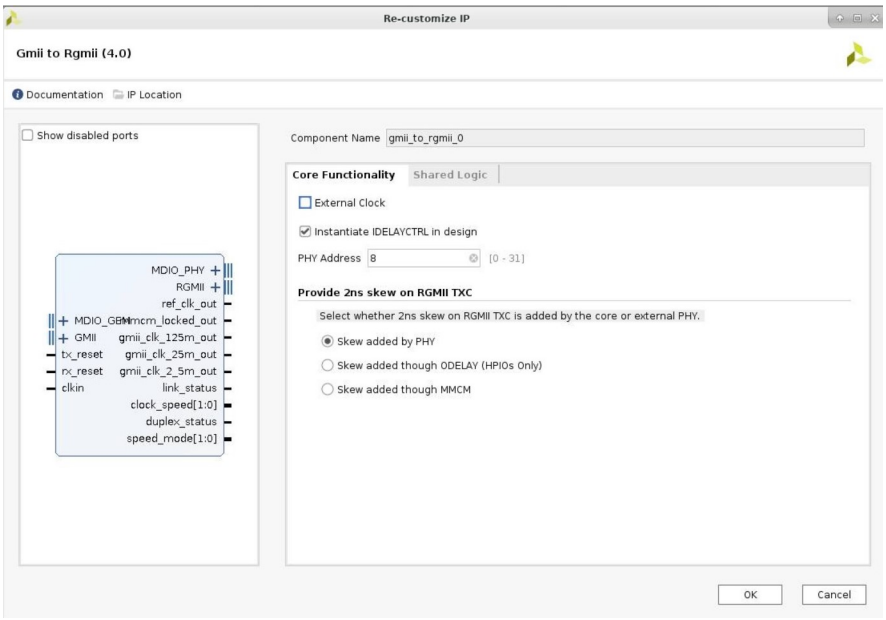

Figure 4.3. Gmii To Rgmii IP block design with relative possibility to add skew.

Since in the project a Linux operative system will be used on the board, no skew will be added by the block (thick on option skew added by PHY). In Linux applications, the simplest way to enable or disable the internal clock delays is through the device tree. The device tree will contain a section for each Ethernet interfaces.

This information is obtained from the demo design, provided by PROfpga and from ETHERNETFMC [32] that, despite describing the different products, were useful to understand the design flow.

In the script created for the automatic creation of the OS system, the following code is added in the device tree, allowing an increase of the delay for the port on both the paths

```
phy@c {
  reg = <0xc>;
  ti,rx-internal-delay = <0x8>;
  ti,tx-internal-delay = <0xa>;
  ti,fifo-depth = <0x1>;
  ti,rxctrl-strap-worka;
};
```

Regarding the constraints and the top VHDL files, only required pins for the GbitEthernet port are left, according to datasheet [21].

In the block diagram several custom settings must be performed, in particular in the Zynq IP block in the I/O configuration, in "high speed" section, GEM0 and MDIO0 are connected to the EMIO port, for instantiating the Ethernet port in the Zynq interface port board, which is directly connected to the PS and does not

need any other block. For the GbitEthernet port of the extension board, GEM3 and MDIO3 are used on MIO pins, being available on the block diagram. These pins are connected to the instantiated Gmii to Rgmii IP block in input, while the MDIO_PHY and RGMII pins exit from the same block. They are connected to relative output pins, created through the "create interface port" procedure by clicking on the right button in the diagram window.

Since the Ethernet port can only be tested with OS system in Vivado SDK, two main Linux application programs are created, respectively a server and a client, in order to establish a TCP communication. These two programs use the: sys/-socket.h, netinet/in.h and netdb.h libraries. The client opens a communication on the specified address IP, provided as parameter, and it stands by until a client asks for a communication on the same IP. If the string "exit" is passed the communication is closed.

At the power on, the IP of the port must be setted, otherwise the communication cannot be established. Some ports might not be listed in the available ones when the ifconfig command is called.

In order to activate a port, this command can be used

```
ip link set dev eth# up
```

Where # indicates the number of Ethernet ports to activate.

If the port is already available, the IP can be setted by typing

```
ip addr add X.X.X.X/16 broadcast + dev eth#
```

In particular after setting the IP of the board, the port is tested firstly with ping command, connecting the board to a PC through the Ethernet. In the second case the board, on which the server program is running, is connected to a raspberry on which the client is running. Later on, programs are swapped. To run the server, the port on which opening it is required, while the client firstly needs the IP address on which the server is running and secondly the port.

A copy operation of a folder is also tested during the communication with raspberry, using the following command on the board

```
scp -r pi@X.X.X.X:/home/pi .
```

All the used files, programs and scheme are available in the appendix C.5.

4.4 DDR4 5 Gbytes ram Board (EB-PDS-DDR4-R6)

Since the compression of the image needs lots of memory, a DDR4 memory is described in this section such as a very important block. This is due to the fact that it

allows to have the largest part of the image ready to be processed. The DDR extension board provides a 5 Gbytes DDR memory, belonging to the MT40A512M16HA-O83E model. In the block diagram (test_ddr) the fundamental block that must be instantiated is the MIG IP block that allows to connects the processor with a user defined memory. In particular the block parameters are set as follows:

- Memory device interface speed: 833 ps;
- Reference input clock speed: 7998 ps;
- Configuration: component,
- Memory part: MT40A512M16HA-O83E;
- Data with: 64 bits.

All others parameters are left as default. This IP is connected to the processor through the AXI bus and its output is connected to the DDR generated port interface. The MIG is a very useful block that allows to generate the memory controllers and the interfaces for the FPGA and to support almost all types of memory, included DDR4. In the constraints and in top VHDL files all pins related to DDR are set as described in datasheet [21].

For this design, only the C code is tested and no OS functions are used. This is due to the fact that the OHB team provides its API, butthe DDR functions call can be implemented as described in the following test.

In Vivado SDK, a standalone code is implemented, in which the base Xilinx writing and reading operations are tested. In particular 8 operations with 8 bits and 4 operations of 16 bits, both in writing mode, are performed, in order to understand how to work with the address. The same approach is used with the read operations, in which 16 operations of 8 bits and 4 operations of 32 bits are performed . All the results are printed on the screen. As usual, all files, configuration and the debug image can be consulted in the appendix C.6.

4.5 Custom AXI slave VHDL implementation controlled by OS

This final design (test_calc) is used to simulate the last step of the thesis, in which the C code must be converted into a VHDL device, that has to interact with the OS as an hardware acceleration. In this design, a simple multiplier is implemented in VHDL, packaged as AXI slave and communicating with the processor through the AXI bus. Subsequently the communication is tested at the first phase as C code with Xilinx functions and with Linux functions by the OS in the following

ones. The constraints and VHDL top files are not different from previous project in which the LEDS are used.

The main idea is similar to the blinker in test performances, where both PS and PL are used. In this case, multiplier components are implemented in VHDL, taking two numbers A and B as inputs on 16 bits and a selector SEL. If the value of the latter at the input is equal to two, the multiplication is calculated and returned to the output on 32 bits port CALC_OUT.

This block is used as a component in the generated code of the AXI slave mymultiplicator_v1_0_S00_AXI at the end of the file. Ports A,B,SEL are respectively connected to the instantiated registers slv_reg0,slv_reg1,slv_reg2, while the port CALC_OUT is connected to the signal calculator_out. Moreover, this last signal is added to the sensitivity list and put at the output instead of the remaining register slv_reg3. In this way only the inner AXI VHDL file is modified and no changes have to be provided to the upper file mymultiplicator_v1_0 of the diagram definition. In fact in the block diagram, once the multiplier is instantiated, it has only the AX, the clock and the reset connection.

In Vivado SDK, two different files are created, the former is the C code for the processor to run in bare metal, while the latter is the C code for Linux system, both communicating and using the multiplier.

For the bare metal application, the code is very similar to the one used in the test of the DDR. In this case, since the instantiated register of the AXI slave is on 32 bits, write operations are based on Xil_Out32 function that writes 32 bits at same times. Since the inputs are on three different registers, each of 32 bits, the address of each operation is shifted by 4 bytes with respect to the base address of the multiplier found in the system.hdf file. The same method is applied to the read operations. This is code is debugged with the print of the values on the screen and with the LEDS blink.

For the Linux application, since the Xilinx functions can not be used, a suitable solution is found in the mmap() Linux system call [33] [34]. This function can be used, by including the sys/mman.h library, to create a mapping from the system physical memory /dev/mem to a defined pointer.

```
page_addr_mult    = (XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR & ~(page_size-1));

page_offset_mult = XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR - page_addr_mult;

ptr_mult =  mmap(NULL, page_size ,PROT_READ|PROT_WRITE,MAP_SHARED, fd ,
            (XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR & ~(page_size-1)));
```

From these commands a memory physical location is created, starting from the address of the multiplier block and occupying a page size of space, that by default is 4096 bytes. This memory location is mapped in read and write modes as shared memory, becoming visible and accessible by all the processes running on the system. After that, read and write operations are performed to the pointer, created with the same offset method used by the bare metal application. As usual, the debug is

performed using the print to the screen.

This is the last design that has been tested, all the built projects are confirmed to be correct by the hardware team OHB, with a delivered and used version. Previous tests are developed in order to understand the functionality of the board and to have a simplified version of the system, on which the compression and the encryption code are tested. All the files produced can be consulted in the appendix C.7.

Chapter 5

Code implementation

Once the hardware system is defined, the compression software can be tested on the board. The first step was to compile the C code of the compression, verifying if there were some problems with the board architecture. The compression and decompression code use the Keccak library that is provided as lib.a static library and it extracts from it the contained files. With the command "ar x libkeccak.a" only objects are obtained without any C code sources. Since the library is compiled for an Ubuntu system x64, it is not compatible with the workstation on which a Centos7 system is running. For this reason, from the github channel of Keccak team [18] all the library is downloaded, including the makefile. Due to the compiler for the board system, the makefile compatible system list is not included. A possible solution is to generate a dynamic library .so for a generic x64 Linux system. After all the files were generated, in order to use the generated files, the compressor and decompressor makefiles of have been modified in this way :

```
//original commands
CCFLAGS = -g -Wall -rdynamic
KeccakFlag = -I ../libkeccak/bin/generic64/libkeccak.a.headers/ -L
    ../libkeccak/bin/generic64/

//modified commands to the generated library
CCFLAGS = -g -Wall -rdynamic -std=c99
KeccakFlag = -I /home/marco.cornelio/Scrivania/CCSDS/main_folder_ccsds/
    XKCP/bin/generic64/libkeccak.a.headers -L
        /home/marco.cornelio/Scrivania/CCSDS/main_folder_ccsds/XKCP/bin/generic64
```

Applying these changes, all the programs provided can be run on the lab workstation. The next step is to run these programs on the boards. Since the only available compiler for the board architecture is on Vivado SDK, all the source files of each program are imported in a Linux application project on SDK. Moreover, also the source files inside the keccak library .so are imported in the project.

This way the compiler has all the information for the executable generations and, after loading the elf file on the SD card and adding the library for use C++ code on the boards (cubito is programmed in C++ language), they all work.

In order to launch the code for the alerts, a single_test bash can be launched.

It runs the test with a dummy pre written alert, otherwise the test bash can be launched, generating 300 random alerts and processing all of them step by step. For testing the compression and encryption algorithms, the test.sh bash can be launched, also using the codec2_NL, in which all the parameters and image path are passed to the compression and decompression programs. After that, the results are passed to cubito, that evaluates all the performance values, comparing the original and the reconstructed images.

Some problems may come up during the running programs, in particular with the compression and decompression tests:

- Function calls used to evaluate the time of the whole compression and decompression chain do not works on the boards. For this reason, they are commented in order to avoid the block of the test;
- Some parameters passed to the compressor by test.sh and codec2_NL.sh are not traduced into the correct value by the optarg function. The incorrect parameters are : weight_initial and weight_final (also on Centos7).

This code implementation is used as back up solution in case VHDL implementation is not possible.

The code that has to be implemented in VHDL is the compressor, while all others programs are used to validate the compression operation. Concerning the alerts, encryption will not be implemented on the board.

In order to implement the code in VHDL, Vivado hls is used, a tool that converts the standard C code functions into VHDL code. After passing all codes and Keccak library sources, the synthesizer is not able to traduce the code, due to the high level functions and to the huge quantities of dynamics parameters with no defined size present in Keccak.

Since it is not considered wise to modify a certificated and validated library, this is settled by implementing by hand Keccak functions used in the code and delegating the remaining part of code to Vivado synthesizer.

5.1 Keccak C code analysis

In the .so library there is not a reference code for understanding the operations performed. For this reason a reverse engineering approach was used, which implies the comparison between the available papers and the code. In Keccak Team web site [35] there are some suggested implementations of Keccak hash function [12].

These codes are the perfect candidates to implement Keccak functions, in particular the Keccak Team has worked on the optimization to facilitate the hardware implementation. For the round functions, only boolean expressions are used, avoiding the involvement of adders or of other complex logic. For this reason, these

designs are very useful when high frequencies are needed in the project, moreover they support all variants (rates, capacities) and use cases, like PRG, for a given lane size, with little changes of VHDL code.

Due to expanse length of the compression code, the alerts program is used for the simulation in C code. It has less code lines and it is mainly focused on the encryption. Despite Keccak functions in these two programs are very similar, the only difference is that the in the alerts code the random numbers generated are taken byte for byte, while in the compression bit per bit.

Starting from the encr.c code, an additional code is derived from it, in which only keccak functions are used in order to create a dataset of input and output to compare with the VHDL code. The code used for the debug can be visualized in appendice.

Since the padding rule is not very clear, some tests are performed to cover all cases and to have a complete dataset to use for VHDL implementation. In particular the following cases are verified and both input and ouput values are printed to file in hexadecimal representation:

- Single key of 4 bytes. Used for a short example of key word;
- Two keys with a small number of bytes. Since in the compression two different keys are used, this is used to simulate the behaviour of the real code using small keys;
- Two keys of 32 bytes. In this case long keys are used like in the original code;
- Two long keys with two fetches. This way it is possible to understand which is the behaviour in case of multiple fetch calls.

An important consideration to make is related to the feed function. Indeed, when it is called, the length parameter of the key must be exactly the length of the key, otherwise if the length is longer, the feed function fills the exceeding part with all zeros and then it adds the termination bit. This means that if the length given as parameter is longer than the key length, the returning output is different with respect to the real key input. All the tests performed can be consulted in the appendice.

5.2 VHDL Keccak implementation

There are three different types of design suggested for the Keccak implementation (version 3.1):

- **High-speed core**: in this version all the Keccak algorithm is implemented in hardware and it operates in a stand-alone version. The core does not use

system resources and the CPU is free to perform others tasks. Moreover to optimize the whole interaction between CPU and Keccak core, a DMA can be used for data transfer. This is recommended if high throughput is required and there is no limit to FPGA space;

- **Low-area coprocessor**: this design is a little different from the previous one. Also here functions are implemented in VHDL, but the state vector is stored in a system memory and only some temporary variables are stored in FPGA registers. This is recommended in systems that do not have large available resources on FPGA;

- **Mid-range core**: there is no information about this design, but from the name it can be assumed that it is a trade-off between the previous two versions.

Since the board used has a huge free space for hardware implementation and the goal is to reach the higher possible working frequency, the high speed core version has been considered as the best solution. The code is composed by 5 different files: keccak,keccak_buffer,keccak_globals,keccak_round and keccak_round_constants_gen.

After the read of a few papers, the structure of the hash function implemented is detected and its behaviour can be summed up in some steps that can be also recognized in the VHDL.

Some restrictions have been put, in order to simplify the implementation. In particular only the Keccak-1600 has been implemented (that is the family used in the project) with constant word length input to 64 bits and parameter $c = 576$.

Keccak is the top file of the design in which all other components are included. In the architecture it is possible to notice that at the reset state all its bits are loaded to zero, after which at the start state the state vector is reset. This repetitive condition is mandatory in the hash function, because after it one cycle of round is computed in order to continue elaborating the hash of the file. THe remaining part of the file is put at the input of the block, but the state must be reset to zero every time a new input block is loaded.

After that the input buffer is completely filled up, which can be considered as the absorbing phase of Keccak algorithm, it starts the permutation evaluation. Once it has finished, the squeezing phase starts, where the data that must be put in output will be returned to the output buffer. In order to save space, the output buffer is not implemented and the data that have to be returned are stored in the input buffer and put in output.

Among all the mapping, it is possible to recognize the signals going out from the state and being XORed with the input r inputs bits. Using $C = 576$ bits means that the capacity part, that remains private and does not change, is on 576 bits, while the rate part is on $r = b - c = 1600 - 576 = 1024$ bits.

In the code these values can be clearly found and they can be represented on the 3D vector.

```
--input mapping
--map part of the state to a vector (256)
i002: for x in 0 to 3 generate
    i003: for i in 0 to 63 generate
        reg_data_vector(64*x+i)<= reg_data(0)(x)(i);
    end generate;
end generate;

--rate part (pubblic) (960)
i10: for row in 0 to 2 generate
    i11: for col in 0 to 4 generate
        i12: for i in 0 to 63 generate
            round_in(row)(col)(i)<= reg_data(row)(col)(i) xor
            (din_buffer_out((row*64*5)+(col*64)+i) and (din_buffer_full and permutation_computed));
        end generate;
    end generate;
end generate;

-- (64)   total rate part 960+64= 1024
i13: for i in 0 to 63 generate
            round_in(3)(0)(i)<= reg_data(3)(0)(i) xor (
            din_buffer_out((3*64*5)+(0*64)+i) and (din_buffer_full and permutation_computed));
end generate;

--capacity part (private) 256
    i01: for col in 1 to 4 generate
        i02: for i in 0 to 63 generate
            round_in(3)(col)(i)<= reg_data(3)(col)(i);
        end generate;
    end generate;

-- (320)   total capacity part 256+320= 576
    i03: for col in 0 to 4 generate
        i04: for i in 0 to 63 generate
            round_in(4)(col)(i)<= reg_data(4)(col)(i);
        end generate;
    end generate;
```

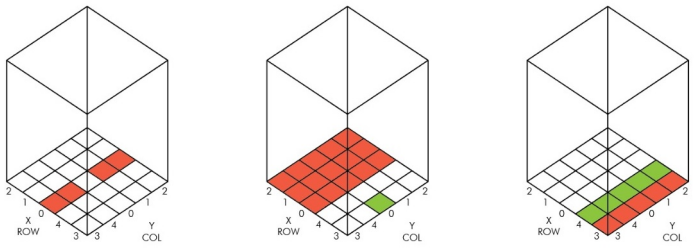

Figure 5.1. On the upside there is the keccak VHDL code, while in the bottom side there are the graphical representations of the taken block in the state vector in order: output,rate and capacity. Red blocks represent the first for cycle, while green the second.

In figure 5.1 the mapping of different parts is displayed, in particular the first block on the left is related to the lanes that are copied in the input part to be returned as output, while the middle block corresponds to the rate part ($r = 1024$). By observing it, it is possible to notice that in the software the mapping is divided into two different for cycles, the first one represented in red, while the second in green. On the right it is represented the state block that displays the lanes used for the capacity part.

In keccak_buffer the definition of the buffer, used as input and output, is provided. The buffer is composed by 1024 locations and ,since the input words are on 64 bits, the buffer is completely filled up after 16 input blocks. The mode variable,

that can be 0 or 1, defines the use of the buffer, being 0 when used as input buffer, while 1 when used as output.

In the input mode, due to the big endian convention used by keccak, a new input block is always inserted in the same position at the higher index (1023 downto 960) and all the old blocks are shifted to lower addresses by 64 bits.

In the output mode, the output vector of 256 bits is loaded in the buffer at the index section (255 downto 0). Since also the output is on 64 bits, only the lower block of the buffer (63 downto 0) is returned as output and the remaining blocks are shifted to the lower index. Since the output vector is on 256 bits, all of them will be returned to the output after 4 operations.

In keccak_globals just the definitions of the 3D state vector are reported, where it is possible to recognize the square base of the cube composed by 5 cubes by 5 cubes.

In keccak_round all the permutations operations are included: theta,ro,pi,chi and iota, implemented using only boolean operations without adders or multipliers in order to reduce the critical path. All the constants used by the iota function are generated in the keccak_round_constants_gen and ,since the required rounds to maintain the security for a state of 1600 are 24, the generated constants are 24.

In figure 5.2 the block diagram of the whole Keccak block hardware is provided. All the signals are reported with their own names, in particular out of the components those referred to keccak file, while inside each block those with the same name used in each component file. The input/output buffer is represented as a block of 16 cubes, which represents the entire buffer length $r = 1024$, subdivided respect to the input block word $block = r/w = 1024/64 = 16$.

In dark grey the input block and the output part of the buffer are highlighted. In particular the former is used as input at the higher index of the buffer, while the latter is used as a storage of the reg_data_vector to put at the output by shifting by one block each time. In the middle of the block there is only the shifting of the new block to the lower index, in order to insert new input blocks until all the buffer is filled.

In the folder of keccak team there are only two test benches with their relatives inputs and expected outputs, in order to verify the code. The most important test bench is the tb_keccak in which the hash function implemented is tested.
The test bench is subdivided into 7 states: INIT, read_first_input, st0, st1, END_HASH1, END_HASH2 and STOP. During the INIT state, the state vector is reset to zero value. In the read_first_input state, the first line of the test file is read, where a number identifier for the corresponding test is provided. Moreover, when the evaluation of the first 1024 bits is finished, the test returns in this state and, if the new character is "-", the input message is terminated, otherwise the hash function with new inputs restarts.

The st0 state can be considered as the absorbing phase of the Keccak function, indeed in this state the input buffer is filled until 16 input blocks are put in the

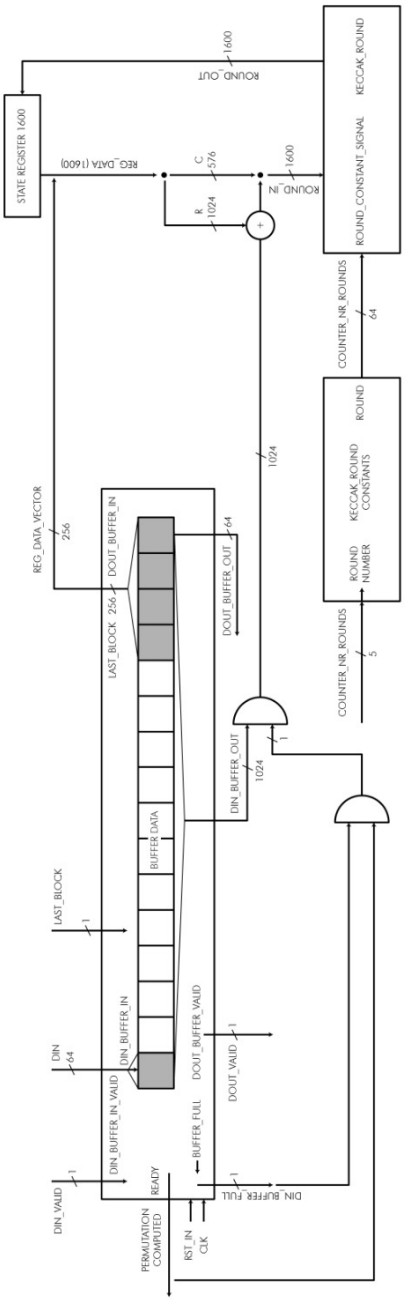

Figure 5.2. Block diagram of Keccak block with all the components and signals used. In dark grey the input block (6a bits) and the output vector (256 bits) sections are highlighted.

buffer, completing the filling.

In the st1 state the 24 permutations are calculated applying the 5 round functions. After all the permutations have been completed, the state switch to the END_HASH1, that is used to change the mode of the buffer from input to output, letting the test entering in the final END_HASH2 state. Here the data picked up from the state vector with the reg_data_vector are stored in the buffer and they are shifted to the lower index part in order to be put at the output. This can be considered the squeezing phase of the Keccak algorithm, where the processed data are pushed out of the block.

As can be seen in the new_test_vector folder, test data are written from right to left, satisfying the big endian convention used internally by Keccak algorithm. Moreover, it is possible to note that all each input stream of 1024 bits ends with number 8. This is the termination bit of the pad10*1 rule, claiming that the input message must be terminated with a bit equal to 1 in little endian convention, which corresponds, in big endian, to terminate each input message with number 8 in hexadecimal (1000 in binary).

In order to have a better view on the working function and on the test, an ASM chart has been drawn and it can be visualized in figure 5.3.

On this code the function PRGsponge can be built, which is used in the compression code.

By analysing different papers [36, 37, 38, 39], it was understood the possibility of implementing a PRG on a sponge function. An hash function and a PRG are quite different. In particular the former takes the input message and ,after receiving the first r bits, it resets the state and continues to elaborate a new block of r bits. The latter, instead, takes as inputs r bits and, if the key is shorter, the message is terminated with 1 bit and the remaining bits to reach r are filled with 0 and it finishes with one bit equal to 1. The Keccak block performs permutation over the input bits and it returns r-8 bits in output. If more bits are requested at the input, a pad message of 1024 bits is put, that is directly XORed with the computed state instead of resetting it.

After this difference has been determined, the modification of the code can be started leaving the Keccak core with rounds functions definition as designed by Keccak team.

The first change is related to the input sequence. Since normally the little endian convention is used, the input was modified in order to satisfy the big endian convention, avoiding the reorganization of the input key. Indeed, the input was adjusted in such a way that the key can be passed to the block in little endian and directly loaded in the buffer in big endian convention.

Since char values are used on 8 bits for the keys, each lane of 64 bits is rearranged in such a way that every single byte of the original line is taken and copied in the same order in the buffer at the lower index. The idea is to work with bytes and to reorder them in the buffer in order to satisfy Keccak convention. The code is

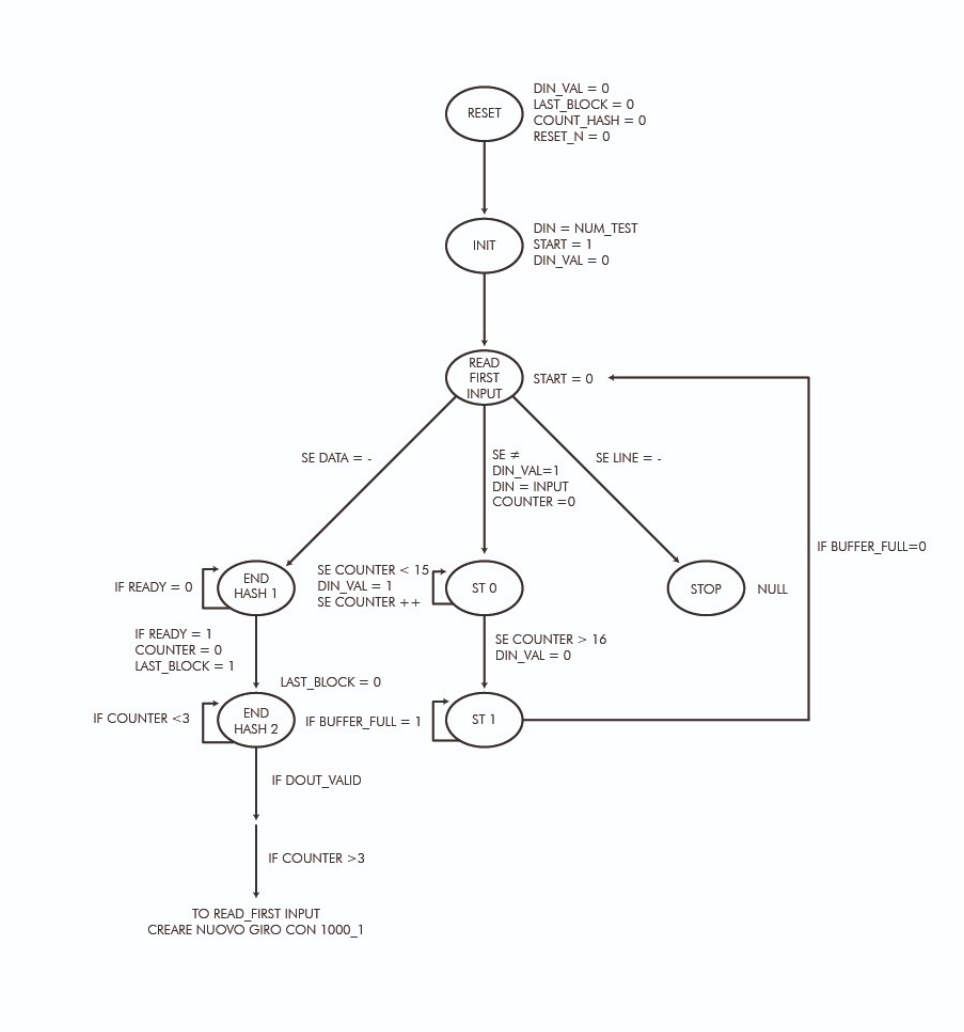

Figure 5.3. ASM chart of the keccak test.

reported below with the aim of providing a more precise description:

```
--buffer_data(1023 downto 960) <= din_buffer_in;
buffer_data(1023 downto 960) <= din_buffer_in(56 to 63) & din_buffer_in(48 to
    55) & din_buffer_in(40 to 47) & din_buffer_in(32 to 39) & din_buffer_in(24
    to 31) & din_buffer_in(16 to 23) & din_buffer_in(8 to 15) & din_buffer_in(0
    to 7);
```

After these changes have been actuated, inputs files have been modified in order to be written in the same way as they are in a common memory register. Indeed, instead of writing them from right to left, they have been written in the normal order from left to right, being compliance with the standards. The termination bit in hexadecimal is changed from 01 to 80, in order to obtain the same termination with the changes.

Another change has been performed on the output modality. Since in hash function only 256 bits are returned in output, they are copied in the buffer pushing 64 bits out per clock cycle. In the PRG the returned bits are 1016, which is consequential to their direct pushed in output, without being copied in the buffer first. For this reason the buffer has been modified, removing the output modality and keeping the input mode intact.

At the input a multiplexer was added which takes the keys as input at the fist permutation cycle and a pad message of 1024 bits in the following cycles when new bits are required.

The input structure of the Keccak block is maintained, namely 64 bits word of the key are copied in the buffer in 16 clock cycles. Consequently all the round functions are performed, each of them maintained as defined by Keccak.

The test bench provided with the code is used as reference code to design a FSM controller, capable to manage each signal going to the keccak block. In such a way the control of the block is very simplified, since it simply uses reset, start and new calculation for the output signals to control it. Also the number of the states are reduced and they are given a more intuitive name. An ASM chart is designed in order to explain the controller operations, provided in figure 5.4.

After run synthesis and implementation, several simulations are performed. All the tests performed with C code are performed again in order to understand the padding rule. Setting the same inputs to the implemented VHDL code, some outputs are obtained, confirming the correctness of the modifications applied to VHDL code. In particular the space on the board used to implement the VHDL code on the FPGA is reported:

- 3205 LUT;
- 2712 FF;
- 0 BRAMs;
- 0 URAms.

Even if the bigger version were chosen among the Keccak suggested, the big available space on the FPGA would make the PRG occupancy negligible. A final test is performed with the keys of 32 bytes used in the code. Performing this test, considering that input structures are left as defined by Keccak and using a clock frequency of 150 MHz equal to the AXI communication bus, the evaluated random bits are available after 627 ns. By reading read 64 bits per clock cycle, as the keccak code provided, all the 1016 bits are read within 841 ns.

The test was also modified by performing the XOR operation between an input message and the random number generated. Assuming that a normal alert is about 10 Kbytes and assuming to read 64 bits per clock cycle, all the file will be encrypted by $66\mu s$. The time measurement on waveform are reported in figure 5.5.

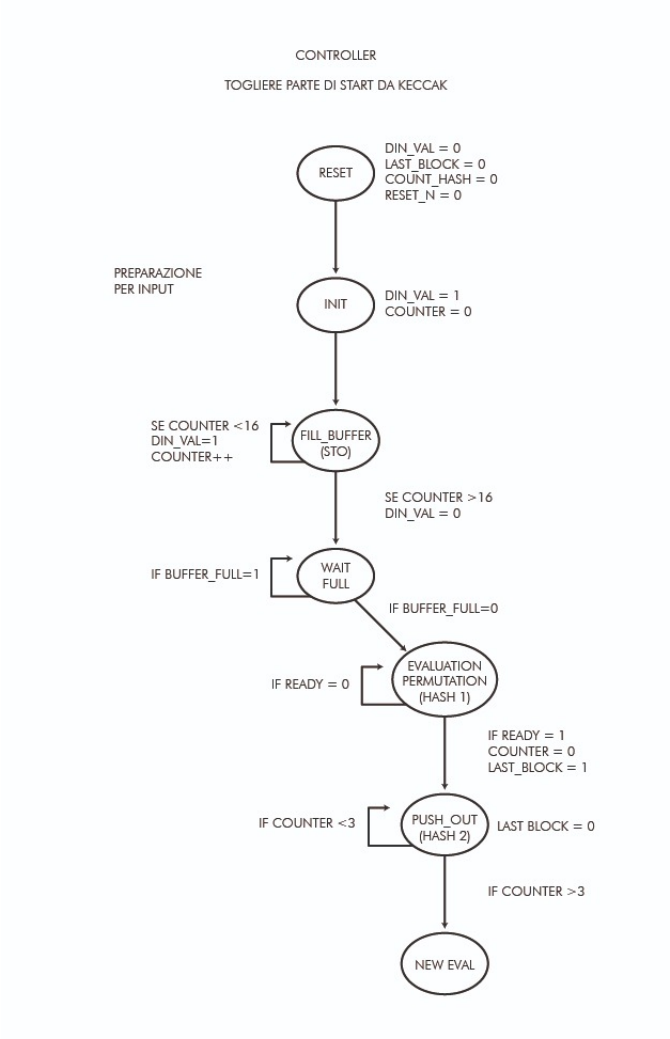

Figure 5.4. ASM chart of the keccak controller.

5.2.1 Compression code implementation

After the Keccak PRG has been implemented by hand, all its functions are commented and the compression code is passed to the synthesizer. The code is still not synthesize. After applying huge modifications to the structure in order to reduce code complexity, a synthesizable version of the block predictor inside the compressor is achieved. It is implemented dividing the whole huge vector of the image input into sub vectors and the whole predictor into multiple stages with one hot coding.

Figure 5.5. Times taken for having valid data at the output and for completing the reading all 1016 output bits.

Chapter 6

Conclusions and future work

This thesis project represents one of the first attempts to implement all the C code of the compression in VHDL code. At the beginning the system architecture was build and tested, which was difficult to perform since the Ultrascale+ family was just announced in 2015 as a early new technology for aerospace and defence purposes. Indeed, Xilinx guide so far is not so completed. Moreover the high cost related to the version has caused the absence of other examples or guides. After the earliest difficulties, it has been possible to realize the custom architecture and to run it. In particular the connection with the DDR4 extension board, The Gbit Ethernet extension board and the interface board are defined. All of them are connected and tested with the motherboard.

A design flow is defined in order to use the processor for controlling and communicating with all the extension boards plugged with the motherboard.

From the ProFPGA example design, a script was written using the petaLinux tool to automatically generate an OS for the board, taking into account all the hardware defined instructions. It was then possible to use all the hardware plugged into the motherboard and to control it through the OS.

Since the C code must be implemented on the FPGA, a test was designed with a custom block implemented in VHDL and loaded on the FPGA. It can be controlled by the OS, considering it such as an hardware accelerator.

Due to the vague guides available on internet, I hope this thesis may be considered as a starting point for everyone who needs to work with the Zynq Ultrascale+ mpsoc board and with the ProFPGA motherboards.

The second part of the thesis instead is focus on the code implementation on the board. Since the code is not understandable by the Vivado synthesizer due to some high level instructions and to the complexity of Keccak library used for the encryption part, it has been decided to reduce the complexity of the code.

The first step was to synthesize the Keccak library. Since it is not consider wise to modify the Keccak library, it has been chosen to modify the VHDL code provided by the Keccak author with a similar function. The compressor uses the

PRG sponge function in order to generate random numbers to encrypt the image, while the VHDL code of Keccak describes the hash function. After some parts of the VHDL code on the hash function have been modified, a PRG function is built using the already implemented rounds permutation functions.

This part was successfully implemented using a relative low space on the FPGA, about 3205 LUT and 2712 FF. The results were also obtained in a short time, making the output bits available after 627 ns.

The C Keccak code has been commented in the compression code, but it still remained not synthesized by the vivado synthetizer. For this reason it was decided to synthesize separately each block of the compressor.

The first block to synthesize was the predictor, then successfully converted into VHDL code.

The thesis work finishes with the forehead mentioned modifications, leaving the remaining ones to the successors.

The future works will consist of implementing all the remaining blocks of the compressor, to integrate also with the PRG hand made code.

Possible improvements can be also applied to the PRG VHDL code, in which the time taken to fill the buffer can be significantly reduced. Instead of using the Keccak suggested method with 64 input bits, supposing to have already memorised the keys, they can be directly stored in the buffer passing from 16 to 1 clock cycle. The same approach can be applied for the message pad after the first permutation, reducing the cycles from 16 to 1. Two further improvements could be the storage of the output bits in a separated buffer and the parallelization of the new computation during the reading of the output bits.

Appendix A

Configuration file generated by ProFPGA

Listing A.1. configuration_file.cfg

```
name = "test_up_down";
created_by = "proFPGA-Builder V1.2.32.1241 (2018C)";
profpga_debug = 0;
debug = 0;
backend = "tcp";
backends :
{
  tcp :
  {
    ipaddr = "169.254.0.2";
    port = 0xD11D;
  };
  pcie :
  {
    device = "/dev/mmi64pcie0";
  };
};
plugin_list = ( "si5338 ProDesign EB-PDS-QSFP+-R1", "dp83865dvh ProDesign
    EB-PDS-GBITETHERNET-R1", "si5338 ProDesign EB-FM-XCZUxxEG-R3",
    "xczuxxeg_eb_config ProDesign EB-FM-XCZUxxEG-R3", "si5338 ProDesign
    EB-PDS-DDR4-R6" );
system_configuration :
{
  sysconfig_match = "FIT";
  fpga_speedgrade_match = "FIT";
  motherboard_1 :
  {
    type = "MB-1M-R2";
    fpga_module_ta1 :
    {
      type = "FM-XCZU19EG-R2";
      speed_grade = "1";
      temp_grade = "E";
      v_io_ba1 = "AUTO";
      v_io_ba2 = "AUTO";
      v_io_bb1 = "AUTO";
      v_io_bb2 = "AUTO";
      v_io_ta2 = "AUTO";
      v_io_tb1 = "AUTO";
```

```
    v_io_tb2 = "AUTO";
    boot_mode = "JTAG";
    ps_npor = "SWITCH";
    ps_nsrst = "SWITCH";
    ps_mgt_lane_0 = "profpga_mgt_03";
    ps_mgt_lane_1 = "profpga_mgt_02";
    ps_mgt_lane_2 = "profpga_mgt_01";
    ps_mgt_lane_3 = "profpga_mgt_00";
  };
  clock_configuration :
  {
    clk_0 :
    {
      source = "LOCAL";
    };
    clk_1 :
    {
      source = "125MHz";
      multiply = 6;
      divide = 15;
    };
    clk_2 :
    {
      source = "125MHz";
      multiply = 5;
      divide = 5;
    };
    clk_3 :
    {
      source = "60MHz";
      multiply = 10;
      divide = 10;
    };
    clk_4 :
    {
      source = "18MHz";
      multiply = 40;
      divide = 40;
    };
    clk_5 :
    {
      source = "125MHz";
      multiply = 6;
      divide = 75;
    };
    clk_6 :
    {
      source = "60MHz";
      multiply = 20;
      divide = 10;
    };
    clk_7 :
    {
      source = "18MHz";
      multiply = 36;
      divide = 18;
    };
  };
  sync_configuration :
  {
    sync_0 :
    {
      source = "GENERATOR";
```

```
      };
      sync_1 :
      {
        source = "GENERATOR";
      };
      sync_2 :
      {
        source = "GENERATOR";
      };
      sync_3 :
      {
        source = "GENERATOR";
      };
      sync_4 :
      {
        source = "GENERATOR";
      };
      sync_5 :
      {
        source = "GENERATOR";
      };
      sync_6 :
      {
        source = "GENERATOR";
      };
      sync_7 :
      {
        source = "GENERATOR";
      };
    };
};
x_board_list = ( "ta2_eb1", "tb1_eb1", "tb2_eb1", "bb1_eb1", "bb2_eb1" );
ta2_eb1 :
{
  type = "BOARD";
  vendor = "ProDesign";
  name = "EB-PDS-QSFP+-R1";
  size = "A1A1";
  positions = ( "motherboard_1.TA2" );
  top_connectors = ( "TA1" );
  v_io_ba1 = "AUTO";
  si5338_registermap_file = "RegisterMap_125MHz.txt";
  si5338_validate_input_clocks_1_2_3 = "yes";
  si5338_validate_input_clocks_4_5_6 = "no";
};
tb1_eb1 :
{
  type = "BOARD";
  vendor = "ProDesign";
  name = "EB-PDS-DEBUG-R1";
  size = "A1A1";
  positions = ( "motherboard_1.TB1" );
  top_connectors = ( );
  v_io_ba1 = "AUTO";
};
tb2_eb1 :
{
  type = "BOARD";
  vendor = "ProDesign";
  name = "EB-PDS-GBITETHERNET-R1";
  size = "A1A1";
  positions = ( "motherboard_1.TB2" );
  top_connectors = ( "TA1" );
```

```
    v_io_ba1 = "AUTO";
    eth_phy1 :
    {
      CLK_MAC_FREQ = 0;
      MAN_MDIX = 0;
      MAC_CLK_EN = 0;
      MDIX_EN = 0;
      MULTI_EN = 0;
      RGMII_SEL0 = 0;
      RGMII_SEL1 = 0;
      PHY_ADDR1 = 0;
      PHY_ADDR2 = 0;
      PHY_ADDR3 = 0;
      PHY_ADDR4 = 0;
      ACT_SPEED0 = 0;
      LNK10_SPEED1 = 0;
      LNK1G_AUTO_NEG = 1;
      LNK100_DUPLEX = 1;
    };
    eth_phy2 :
    {
      CLK_MAC_FREQ = 0;
      MAN_MDIX = 0;
      MAC_CLK_EN = 0;
      MDIX_EN = 0;
      MULTI_EN = 0;
      RGMII_SEL0 = 0;
      RGMII_SEL1 = 0;
      PHY_ADDR1 = 0;
      PHY_ADDR2 = 0;
      PHY_ADDR3 = 0;
      PHY_ADDR4 = 0;
      ACT_SPEED0 = 0;
      LNK10_SPEED1 = 0;
      LNK1G_AUTO_NEG = 1;
      LNK100_DUPLEX = 1;
    };
  };
  bb1_eb1 :
  {
    type = "BOARD";
    vendor = "ProDesign";
    name = "EB-FM-XCZUxxEG-R3";
    size = "A1A1";
    positions = ( "motherboard_1.BB1" );
    top_connectors = ( );
    v_io_ba1 = "AUTO";
    si5338_registermap_file = "RegisterMap_26_27_125_26_MHz.txt";
    si5338_validate_input_clocks_1_2_3 = "yes";
    si5338_validate_input_clocks_4_5_6 = "no";
    gpio_expander1 :
    {
      usb_id_sel = 0;
      usb_cvbus_sel = 1;
      usb_hd_mode_sel0 = 1;
      usb_hd_mode_sel1 = 1;
      gem3_exp_reset_n = 1;
    };
  };
  bb2_eb1 :
  {
    type = "BOARD";
    vendor = "ProDesign";
```

```
    name = "EB-PDS-DDR4-R6";
    size = "A1A1";
    positions = ( "motherboard_1.BB2" );
    top_connectors = ( );
    v_io_ba1 = "AUTO";
    si5338_registermap_file = "RegisterMap_125MHz.txt";
    si5338_validate_input_clocks_1_2_3 = "yes";
    si5338_validate_input_clocks_4_5_6 = "no";
  };
};
```

Appendix B

Script for automatic generation

B.1 Tcl file that instantiate Zynq IP block in Vivado block diagram

Listing B.1. vivado.tcl

```
# ============================================================================
#  IMPORTANT: Pro Design Confidential (Internal Use Only)
#  COPYRIGHT (C) 2018, Pro Design Electronic GmbH
#
#  THIS FILE MAY NOT BE MODIFIED, DISCLOSED, COPIED OR DISTRIBUTED WITHOUT THE
#  EXPRESSED WRITTEN CONSENT OF PRO DESIGN.
#
#  Pro Design Electronic GmbH              http://www.prodesign-europe.com
#  Albert-Mayer-Strasse 14-16              info@prodesign-europe.com
#  83052 Bruckmuehl                        +49 (0)8062 / 808 - 0
#  Germany
# ============================================================================
#  Project        : ProDesign proFPGA
#  Module         : PROF119A tcl file
#! @brief         Script to generate the Vivado Project for PROF119A/120A
#! @file          build.sh
#! @author        Marko Salzmann <Marko.Salzmann@prodesign-europe.com>
#! @date          2018-01-08
#! @version       0.1
#! @version       0.2 (update for vivado 2018.1)
#! @copyright     2018 (C) Pro Design Electronic GmbH
#!
# ============================================================================


# save current directory
#set curdir [pwd]

# set project vars
#set prjname "PROF-FM-XCZU1xEG-1"
#set prjnamefol "/PROF-FM-XCZU1xEG-1/"

# set install path
#set prjfolder "$curdir$prjnamefol"

# create vivado project
```

```
#create_project $prjname $prjfolder -part xczu19eg-ffvb1517-1-e
#set_property coreContainer.enable 1 [current_project]
#create_bd_design "design_1"

# add processing system
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:zynq_ultra_ps_e:3.2 zynq_ultra_ps_e_0
endgroup

# connect clk
connect_bd_net [get_bd_pins zynq_ultra_ps_e_0/maxihpm0_lpd_aclk] [get_bd_pins
    zynq_ultra_ps_e_0/pl_clk0]

# add interfaces
startgroup
set_property -dict [list CONFIG.PSU__CAN0__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__CAN0__PERIPHERAL__IO {MIO 46 .. 47}
    CONFIG.PSU__DPAUX__PERIPHERAL__IO {MIO 27 .. 30}
    CONFIG.PSU__ENET3__PERIPHERAL__ENABLE {1} CONFIG.PSU__ENET3__GRP_MDIO__ENABLE
    {1} CONFIG.PSU__GPIO1_MIO__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__I2C1__PERIPHERAL__ENABLE {1} CONFIG.PSU__I2C1__PERIPHERAL__IO
    {MIO 48 .. 49} CONFIG.PSU__SATA__REF_CLK_FREQ {125} CONFIG.PSU__DP__LANE_SEL
    {Dual Lower} CONFIG.PSU__SD0__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__SD0__PERIPHERAL__IO {MIO 13 .. 16 21 22}
    CONFIG.PSU__SD0__GRP_CD__ENABLE {1} CONFIG.PSU__SD0__SLOT_TYPE {SD 2.0}
    CONFIG.PSU__UART1__PERIPHERAL__ENABLE {1} CONFIG.PSU__UART1__PERIPHERAL__IO
    {MIO 8 .. 9} CONFIG.PSU__USB0__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__USB3_0__PERIPHERAL__ENABLE {1} CONFIG.PSU__USB3_0__PERIPHERAL__IO
    {GT Lane2} CONFIG.PSU__DISPLAYPORT__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__SATA__PERIPHERAL__ENABLE {1} CONFIG.PSU__SATA__LANE1__IO {GT
    Lane3} CONFIG.PSU__CRF_APB__DP_VIDEO_REF_CTRL__SRCSEL {VPLL}
    CONFIG.PSU__CRF_APB__DP_AUDIO_REF_CTRL__SRCSEL {DPLL}
    CONFIG.PSU__CRF_APB__DP_STC_REF_CTRL__SRCSEL {DPLL}
    CONFIG.PSU__CRF_APB__TOPSW_MAIN_CTRL__SRCSEL {DPLL}
    CONFIG.PSU__CRL_APB__SDIO0_REF_CTRL__FREQMHZ {50}] [get_bd_cells
    zynq_ultra_ps_e_0]
endgroup

# set voltages to 1,8V and 3,3V
startgroup
set_property -dict [list CONFIG.PSU_BANK_0_IO_STANDARD {LVCMOS33}
    CONFIG.PSU_BANK_1_IO_STANDARD {LVCMOS18} CONFIG.PSU_BANK_2_IO_STANDARD
    {LVCMOS18} CONFIG.PSU_BANK_3_IO_STANDARD {LVCMOS18}] [get_bd_cells
    zynq_ultra_ps_e_0]
endgroup

# setup ddr4 memmory
startgroup
set_property -dict [list CONFIG.SUBPRESET1 {DDR4_KINGSTON_KVR21SE15S8}]
    [get_bd_cells zynq_ultra_ps_e_0]
endgroup

# set sdio clk to slew
startgroup
set_property -dict [list CONFIG.PSU_MIO_22_SLEW {fast}] [get_bd_cells
    zynq_ultra_ps_e_0]
endgroup

# add qspi flash
startgroup
set_property -dict [list CONFIG.PSU__QSPI__PERIPHERAL__ENABLE {1}
    CONFIG.PSU__QSPI__PERIPHERAL__DATA_MODE {x4}] [get_bd_cells zynq_ultra_ps_e_0]
endgroup
```

```
# generate top level design wrapper
#make_wrapper -files [get_files
    ${curdir}${prjnamefol}${prjname}.srcs/sources_1/bd/design_1/design_1.bd] -top
#add_files -norecurse
    ${curdir}${prjnamefol}${prjname}.srcs/sources_1/bd/design_1/hdl/design_1_wrapper.v
#update_compile_order -fileset sources_1

# generate block design
#generate_target all [get_files
    ${curdir}${prjnamefol}${prjname}.srcs/sources_1/bd/design_1/design_1.bd]

# save design
#save_bd_design

# export hardware description file
#file mkdir ${curdir}${prjnamefol}${prjname}.sdk
#write_hwdef -force  -file
    ${curdir}${prjnamefol}${prjname}.sdk/design_1_wrapper.hdf
```

B.2 Bash for petalinux to create OS system for the Zynq board

Listing B.2. build_mod.sh

```
#!/bin/bash -x

# Define usage function
function PrintUsage() {
 echo "-------------------------------------------------------------------"
 echo "---- Build script for PROF-FM-XCZU1xEG-1 demo -----"
 echo "-------------------------------------------------------------------"
 echo "  USAGE:     ./build.sh"
 echo "  USAGE:     ./build.sh clean"
 echo " "
 echo "  EXAMPLES: ./build.sh       - Build Vivado and XSDK project for
     PROF-FM-XCZU1xEG-1"
 echo "  EXAMPLES: ./build.sh clean - cleanup previously build"
 echo " "
 exit 1
}

source /software/scripts/init_vivado


# abort on error
set -e

# color declaration
RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m'



# check if petalinux is sourced
if [ -z $PETALINUX_VER ] ; then
  echo -e "${RED}ERROR:${NC} Variable PETALINUX_VER is undefined! Did you setup
      Petalinux correctly?"
```

```
  exit
elif [[ $PETALINUX_VER != '2018.3' ]] ; then
  echo -e "${RED}ERROR:${NC} This script requires Petalinux version 2018.1."
  echo "Petalinux location: ` which petalinux`"
  echo "Petalinux version: `petalinux -version`"
  exit
fi

#seach path file
for i in $( find . -maxdepth 1 -name *.sdk -printf %P)
do
  echo "yes"
  SDK_PATH=$i
  echo "$SDK_PATH"
done

for j in $( find ./$SDK_PATH -maxdepth 1 -name *.hdf -printf %P)
do
  echo "ok"
  WRAPPER_NAME=$j
  echo "$WRAPPER_NAME"
done



# creating petalinux project
if [ -r ./$SDK_PATH/$WRAPPER_NAME ]; then
  echo "Creating ZYNQMP Petalinux Project."
  petalinux-create -t project --name TEST_PETA --template zynqMP
  if [ -d ./TEST_PETA ]; then
    echo "Using preconfigured config file."
    cp ./source/config ./TEST_PETA/project-spec/configs
    cp ./source/rootfs_config ./TEST_PETA/project-spec/configs
    echo "Getting hardware information from hdf-file."
    petalinux-config --get-hw-description ./$SDK_PATH -p ./TEST_PETA --oldconfig
    #petalinux-config --get-hw-description ./$SDK_PATH -p ./TEST_PETA
    #aggiunto per kernel
    echo "create app client"
    petalinux-create -t apps --name client --enable -p ./TEST_PETA
    echo "substitute file c"
    rm -f ./TEST_PETA/project-spec/meta-user/recipes-apps/client/files/client.c
    cp ./$SDK_PATH/client/src/client.c
        ./TEST_PETA/project-spec/meta-user/recipes-apps/client/files
    echo "create app server"
    petalinux-create -t apps --name server --enable -p ./TEST_PETA
    echo "substitute file c"
    rm -f ./TEST_PETA/project-spec/meta-user/recipes-apps/server/files/server.c
    cp ./$SDK_PATH/server/src/server.c
        ./TEST_PETA/project-spec/meta-user/recipes-apps/server/files
    echo "Loading kernel config"
    petalinux-config -c kernel -p ./TEST_PETA
    echo "Loading preconfigured rootfs config"
    petalinux-config -c rootfs --oldconfig -p ./TEST_PETA
    #aggiunto per mettere librerie in piu'
    petalinux-config -c rootfs -p ./TEST_PETA
    #aggiunto per u-boot
    echo "Loading device-tree config"
    petalinux-config -c device-tree -p ./TEST_PETA
    echo "Building Petalinux for ZYNQMP."
    if petalinux-build -p ./TEST_PETA ; then echo -e "Built finished
        successfully!"; else echo -e "${RED}ERROR:${NC} Built failed. Check
        Petalinux log file for further information!"; exit 1; fi
  fi
```

```
else
  echo "${RED}ERROR:${NC}
      ./PROF-FM-XCZU1xEG-1/PROF-FM-XCZU1xEG-1.sdk/design_1_wrapper.hdf no such
      file or directory!"
  echo "                  Can't create petalinux project without hardware
      description file!"
  exit 1
fi
# creating sdcard directory and copy all needed files into
if [ -d ./TEST_PETA/images/linux ]; then

  #petalinux-package --boot --format BIN --fsbl
      TEST_PETA/images/linux/zynqmp_fsbl.elf --u-boot
      TEST_PETA/images/linux/u-boot.elf --pmufw TEST_PETA/images/linux/pmufw.elf
      --fpga TEST_PETA/images/linux/*.bit --force -p ./TEST_PETA

  echo "Creating sdcard directory and copy all needed files."
  mkdir -p ./sdcard
  # copy petalinux files to sdcard
  #cp ./TEST_PETA/images/linux/BOOT.BIN      ./sdcard
  cp ./TEST_PETA/images/linux/image.ub       ./sdcard
  cp ./TEST_PETA/images/linux/system.dtb     ./sdcard
  cp ./TEST_PETA/images/linux/rootfs.cpio.gz.u-boot ./sdcard/uramdisk.image.gz
  cp ./source/uEnv_sdio_initramfs.txt      ./sdcard/uEnv.txt

  # make Boot.bin
  bootgen -arch zynqmp -image ./source/output.bif -o ./sdcard/BOOT.BIN -w

fi



if [ -r ./sdcard/system.dtb ]; then
  # fixing devicetree for USB Host mode
  echo "Converting devicetree binary to devicetree source"
  if dtc -I dtb -O dts ./sdcard/system.dtb > ./sdcard/system.dts; then echo
      "done"; else echo "failed to converting devicetree. Is package dtc
      installed?"; exit 1; fi
  echo "Adding usb host mode to devicetree"
  sed -i 's/dwc3@fe200000/dwc3_0: dwc3@fe200000/g' ./sdcard/system.dts
  echo '&dwc3_0 {'             >> ./sdcard/system.dts
  echo '    dr_mode = "host";' >> ./sdcard/system.dts
  echo '};'                    >> ./sdcard/system.dts
  echo "Adding ethernet phy to devicetree source"
  sed -i '/ethernet@ff0e0000/{N;N;N;N;N;N;N;N;N;N;N;N;N;N;N;
  r ./source/add_phy
  }' ./sdcard/system.dts
  echo "Deactivating write protection pin"
  sed -i '/sdhci@ff160000/{N;N;N;N;N;N;N;N;N;N;N;N;N;N;N;
  r ./source/remove_wp
  }' ./sdcard/system.dts
  echo "Deactivating UHS Mode"
  sed -i '/sdhci@ff160000/{N;N;N;N;N;N;N;N;N;N;N;N;N;N;N;
  r ./source/remove_uhs
  }' ./sdcard/system.dts
  echo "Rebuild fixed dtb and deleting temp file."
  dtc -I dts -O dtb ./sdcard/system.dts > ./sdcard/system.dtb
  rm -f ./sdcard/system.dts
  echo "Build finished successfully."
  echo ""
  echo "Copy entiry content of ./sdcard to an empty FAT formatted SD-Card and
      boot the system."
```

```
  echo "    - source
      /opt/prodesign/profpga/proFPGA-<release_version>/bin/settings64*.sh"
  echo "    - change system.cfg settings to your needs."
  echo "    - run: profpga_run system.cfg -u"
  echo ""
else
  echo -e "${RED}ERROR:${NC} ./sdcard/system.dtb - no such file or directory!
      Exiting."
  exit 1
fi
```

Appendix C

Base tests

C.1 Only FPGA test files

Listing C.1. mb_1_TA1.xdc constraints file

```
# FPGA module FM-XCZU19EG-R2
# pins which are not connected to an x-board
set_property  PACKAGE_PIN H28 [get_ports {CLK_N_0}]
##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVDS [get_ports {CLK_N_0}]
##############################################################

##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVDS [get_ports {CLK_P_0}]
set_property  DIFF_TERM_ADV TERM_100 [get_ports {CLK_P_0}]
create_clock -name CLK -period 10 [get_ports {CLK_P_0}]
##############################################################

##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_GREEN}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_RED}]
##############################################################
set_property  PACKAGE_PIN F27 [get_ports {LED_GREEN}]
set_property  PACKAGE_PIN C26 [get_ports {LED_RED}]


# VREF/DCI constraints for banks related to connector BB2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 72]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 73]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 74]

# VREF/DCI constraints for banks related to connector TA2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 66]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 64]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 65]

# VREF/DCI constraints for banks related to connector TB1
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 67]
```

```
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 69]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 68]
```

Listing C.2. mb_1_TA1.vhd TOP file

```
-- FPGA module FM-XCZU19EG-R2

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Library UNISIM;
use UNISIM.vcomponents.all;

entity mb_1_TA1 is
  port (
   -- pins which are not connected to an x-board
   CLK_N_0 : in std_logic;
   CLK_P_0 : in std_logic;
   LED_GREEN : out std_logic;
   LED_RED : out std_logic

  );
end entity mb_1_TA1;

architecture beh of mb_1_TA1 is

  signal CLK : std_logic;
  signal cnt : std_logic_vector(31 downto 0);
  signal LED_GREEN_int : std_logic;

begin

  ------------------------------------------------------------------------------
  -- Input differential buffer UD001-3.19-Hardware-UserManual.docx p. 30
  ------------------------------------------------------------------------------

  -- Begin of IBUFGDS_inst instantiation

  IBUFGDS_inst : IBUFGDS
    generic map(
      DIFF_TERM    => true,             -- Differential Termination
      IBUF_LOW_PWR => true,  -- Low power (TRUE) vs. performance (FALSE) setting
      -- for referenced I/O standards
      IOSTANDARD   => "LVDS")
    port map (
      O  => CLK,                                -- Clock buffer output
      I  => CLK_P_0,  -- Diff_p clock buffer input (connect directly to top-level
          port)
      IB => CLK_N_0  -- Diff_n clock buffer input (connect directly to top-level
          port)
    );

  -- End of IBUFGDS_inst instantiation

  process (CLK) is
  begin  -- process
    if CLK'event and CLK = '1' then  -- rising clock edge
      if (cnt(27) = '1') then
        cnt <= (others => '0');
      else
```

```
        cnt <= cnt + '1';
      end if;
    end if;
  end process;

  process (CLK) is
  begin  -- process
    if CLK'event and CLK = '1' then  -- rising clock edge
      if (cnt(27) = '1') then
        LED_GREEN_int <= not(LED_GREEN_int);
      end if;
    end if;
  end process;

  LED_GREEN <= LED_GREEN_int;
  LED_RED <= not(LED_GREEN_int);

end beh;
```

C.2 Processor and FPGA first design

Listing C.3. mb_1_TA1.xdc constraints file

```
# (c) Copyright 2012-2017 PRO DESIGN Electronic GmbH.
# All rights reserved.
#
# This file is owned and controlled by ProDesign and must be used solely
# for design, simulation, implementation and creation of design files
# limited to profpga systems or technologies. Use with non-profpga
# systems or technologies is expressly prohibited and immediately
# terminates your license.
#
# PRODESIGN IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" SOLELY
# FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR PRODESIGN SYSTEMS. BY
# PROVIDING THIS DESIGN, CODE, OR INFORMATION AS ONE POSSIBLE
# IMPLEMENTATION OF THIS FEATURE, APPLICATION OR STANDARD, PRODESIGN IS
# MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION IS FREE FROM ANY
# CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE FOR OBTAINING ANY
# RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION. PRODESIGN EXPRESSLY
# DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
# IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
# REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
# INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
# PARTICULAR PURPOSE.
#
# ProDesign products are not intended for use in life support appliances,
# devices, or systems. Use in such applications are expressly
# prohibited.
#
# This file was generated by profpga_brdgen version 9.02
#   on Thu Jan 31 14:27:01 2019


# FPGA module FM-XCZU19EG-R2
# pins which are not connected to an x-board
set_property  PACKAGE_PIN H28 [get_ports {CLK_N_0}]
############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVDS [get_ports {CLK_N_0}]
############################################################
```

```
set_property  PACKAGE_PIN J27 [get_ports {CLK_P_0}]
##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVDS [get_ports {CLK_P_0}]
set_property  DIFF_TERM_ADV TERM_100 [get_ports {CLK_P_0}]
create_clock -name CLK -period 10 [get_ports {CLK_P_0}]
##############################################################

set_property  PACKAGE_PIN A23 [get_ports {LED_BLUE}]
set_property  PACKAGE_PIN J20 [get_ports {LED_BLUE2}]
##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_BLUE}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_BLUE2}]
##############################################################
set_property  PACKAGE_PIN F27 [get_ports {LED_GREEN}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_GREEN}]
set_property  PACKAGE_PIN E20 [get_ports {LED_GREEN2}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_GREEN2}]
set_property  PACKAGE_PIN C26 [get_ports {LED_RED}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_RED}]
set_property  PACKAGE_PIN D20 [get_ports {LED_RED2}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_RED2}]
set_property  PACKAGE_PIN A22 [get_ports {LED_YELLOW}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_YELLOW}]
set_property  PACKAGE_PIN M28 [get_ports {LED_YELLOW2}]
set_property  IOSTANDARD LVCMOS18 [get_ports {LED_YELLOW2}]



# VREF/DCI constraints for banks related to connector BB2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 72]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 73]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 74]

# VREF/DCI constraints for banks related to connector TA2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 66]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 64]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 65]

# VREF/DCI constraints for banks related to connector TB1
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 67]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 69]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 68]
```

Listing C.4. TOP.vhd TOP file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Library UNISIM;
use UNISIM.vcomponents.all;


entity TOP_level is
port (
   LED_BLUE : out std_logic;
   LED_BLUE2 : out std_logic;
   LED_GREEN : out std_logic;
   LED_GREEN2 : out std_logic;
```

```
    LED_RED : out std_logic;
    LED_RED2 : out std_logic;
    LED_YELLOW : out std_logic;
    LED_YELLOW2 : out std_logic

    );
end TOP_level;

architecture Behavioral of TOP_level is

component edt_xczu19eg_wrapper is
  port (
    gpio_rtl_0_tri_o : out STD_LOGIC_VECTOR ( 7 downto 0 )
  );
end component;

signal LED_strip : std_logic_vector (7 downto 0);

begin

    processore : edt_xczu19eg_wrapper port map (gpio_rtl_0_tri_o => LED_strip);

   LED_BLUE <= LED_strip(3);
   LED_GREEN <= LED_strip(0);
   LED_RED <= LED_strip(1);
   LED_YELLOW <= LED_strip(2);
   LED_BLUE2 <= LED_strip(7);
   LED_GREEN2 <= LED_strip(4);
   LED_RED2 <= LED_strip(5);
   LED_YELLOW2 <= LED_strip(6);


end Behavioral;
```

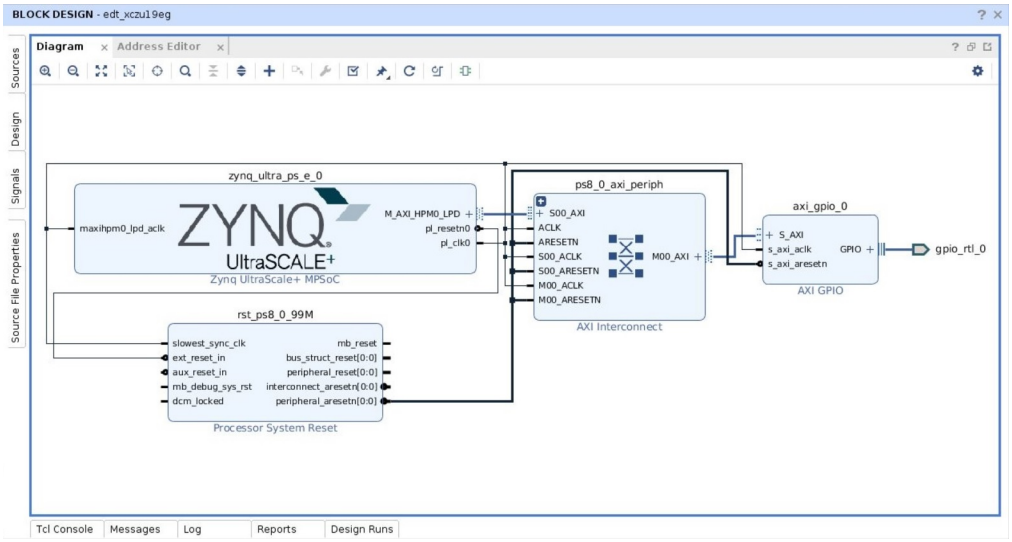

Figure C.1. Block diagram of the test_hello design.

Listing C.5. helloworld.c C program

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <xgpio.h>
#include "xil_io.h"
#include "xparameters.h"
#include "sleep.h"


int main()
{

  XGpio led_blinker;

  int cont=0,turn=0;

  //initialize output xgpio variable
  XGpio_Initialize(&led_blinker,XPAR_AXI_GPIO_0_DEVICE_ID);

  //set first channel buffer tristate to output
  XGpio_SetDataDirection(&led_blinker,1,0x0);

  init_platform();

  while(cont<100){

    if(turn==0){

      XGpio_DiscreteWrite(&led_blinker,1,0b11110000);
      print("hello TOP side\n\r");
      printf("printf normale TOP\n\r");
      turn=1;
    }else{

      XGpio_DiscreteWrite(&led_blinker,1,0b00001111);
      xil_printf("hello DOWN side\n\r");
      turn=0;
    }


    usleep(2000000);
    cont++;
  }
    cleanup_platform();
    return 0;
}
```

C.3 Processor and FPGA second design

Listing C.6. blinker_v1_0.vhd IP description file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity blinker_v1_0 is
  generic (
    -- Users to add parameters here
```

```
    -- User parameters ends
    -- Do not modify the parameters beyond this line


    -- Parameters of Axi Slave Bus Interface S00_AXI
    C_S00_AXI_DATA_WIDTH  : integer := 32;
    C_S00_AXI_ADDR_WIDTH  : integer := 4
  );
  port (
    -- Users to add ports here
    --added to put in output led_string
    output_led : out std_logic_vector(7 downto 0);

    -- User ports ends
    -- Do not modify the ports beyond this line


    -- Ports of Axi Slave Bus Interface S00_AXI
    s00_axi_aclk  : in std_logic;
    s00_axi_aresetn : in std_logic;
    s00_axi_awaddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
    s00_axi_awprot  : in std_logic_vector(2 downto 0);
    s00_axi_awvalid : in std_logic;
    s00_axi_awready : out std_logic;
    s00_axi_wdata : in std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    s00_axi_wstrb : in std_logic_vector((C_S00_AXI_DATA_WIDTH/8)-1 downto 0);
    s00_axi_wvalid  : in std_logic;
    s00_axi_wready  : out std_logic;
    s00_axi_bresp : out std_logic_vector(1 downto 0);
    s00_axi_bvalid  : out std_logic;
    s00_axi_bready  : in std_logic;
    s00_axi_araddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
    s00_axi_arprot  : in std_logic_vector(2 downto 0);
    s00_axi_arvalid : in std_logic;
    s00_axi_arready : out std_logic;
    s00_axi_rdata : out std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    s00_axi_rresp : out std_logic_vector(1 downto 0);
    s00_axi_rvalid  : out std_logic;
    s00_axi_rready  : in std_logic
  );
end blinker_v1_0;

architecture arch_imp of blinker_v1_0 is

  -- component declaration
  component blinker_v1_0_S00_AXI is
    generic (
    C_S_AXI_DATA_WIDTH  : integer := 32;
    C_S_AXI_ADDR_WIDTH  : integer := 4
    );
    port (
    --added to port declaration
        OUTPUT_LED : out std_logic_vector(7 downto 0);
    S_AXI_ACLK  : in std_logic;
    S_AXI_ARESETN : in std_logic;
    S_AXI_AWADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_AWPROT  : in std_logic_vector(2 downto 0);
    S_AXI_AWVALID : in std_logic;
    S_AXI_AWREADY : out std_logic;
    S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
    S_AXI_WVALID  : in std_logic;
```

```
    S_AXI_WREADY  : out std_logic;
    S_AXI_BRESP : out std_logic_vector(1 downto 0);
    S_AXI_BVALID  : out std_logic;
    S_AXI_BREADY  : in std_logic;
    S_AXI_ARADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_ARPROT  : in std_logic_vector(2 downto 0);
    S_AXI_ARVALID : in std_logic;
    S_AXI_ARREADY : out std_logic;
    S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_RRESP : out std_logic_vector(1 downto 0);
    S_AXI_RVALID  : out std_logic;
    S_AXI_RREADY  : in std_logic
    );
  end component blinker_v1_0_S00_AXI;

begin

-- Instantiation of Axi Bus Interface S00_AXI
blinker_v1_0_S00_AXI_inst : blinker_v1_0_S00_AXI
  generic map (
    C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
    C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
  )
  port map (
    S_AXI_ACLK  => s00_axi_aclk,
    S_AXI_ARESETN => s00_axi_aresetn,
    S_AXI_AWADDR  => s00_axi_awaddr,
    S_AXI_AWPROT  => s00_axi_awprot,
    S_AXI_AWVALID => s00_axi_awvalid,
    S_AXI_AWREADY => s00_axi_awready,
    S_AXI_WDATA => s00_axi_wdata,
    S_AXI_WSTRB => s00_axi_wstrb,
    S_AXI_WVALID  => s00_axi_wvalid,
    S_AXI_WREADY  => s00_axi_wready,
    S_AXI_BRESP => s00_axi_bresp,
    S_AXI_BVALID  => s00_axi_bvalid,
    S_AXI_BREADY  => s00_axi_bready,
    S_AXI_ARADDR  => s00_axi_araddr,
    S_AXI_ARPROT  => s00_axi_arprot,
    S_AXI_ARVALID => s00_axi_arvalid,
    S_AXI_ARREADY => s00_axi_arready,
    S_AXI_RDATA => s00_axi_rdata,
    S_AXI_RRESP => s00_axi_rresp,
    S_AXI_RVALID  => s00_axi_rvalid,
    S_AXI_RREADY  => s00_axi_rready,
    --added to the port map
        OUTPUT_LED => output_led

  );

  -- Add user logic here

  -- User logic ends

end arch_imp;
```

Listing C.7. blinker_v1_0_S00_AXI slave implementation file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity blinker_v1_0_S00_AXI is
```

```
generic (
  -- Users to add parameters here

  -- User parameters ends
  -- Do not modify the parameters beyond this line

  -- Width of S_AXI data bus
  C_S_AXI_DATA_WIDTH  : integer := 32;
  -- Width of S_AXI address bus
  C_S_AXI_ADDR_WIDTH  : integer := 4
);
port (
  -- Users to add ports here
  --##################################################
  --aggiunto qui
      --INPUT_LED_STRIP : in std_logic_vector(7 downto 0);
      OUTPUT_LED : out std_logic_vector(7 downto 0);
      --##################################################
  -- User ports ends
  -- Do not modify the ports beyond this line

  -- Global Clock Signal
  S_AXI_ACLK  : in std_logic;
  -- Global Reset Signal. This Signal is Active LOW
  S_AXI_ARESETN : in std_logic;
  -- Write address (issued by master, acceped by Slave)
  S_AXI_AWADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  -- Write channel Protection type. This signal indicates the
      -- privilege and security level of the transaction, and whether
      -- the transaction is a data access or an instruction access.
  S_AXI_AWPROT  : in std_logic_vector(2 downto 0);
  -- Write address valid. This signal indicates that the master signaling
      -- valid write address and control information.
  S_AXI_AWVALID : in std_logic;
  -- Write address ready. This signal indicates that the slave is ready
      -- to accept an address and associated control signals.
  S_AXI_AWREADY : out std_logic;
  -- Write data (issued by master, acceped by Slave)
  S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  -- Write strobes. This signal indicates which byte lanes hold
      -- valid data. There is one write strobe bit for each eight
      -- bits of the write data bus.
  S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
  -- Write valid. This signal indicates that valid write
      -- data and strobes are available.
  S_AXI_WVALID  : in std_logic;
  -- Write ready. This signal indicates that the slave
      -- can accept the write data.
  S_AXI_WREADY  : out std_logic;
  -- Write response. This signal indicates the status
      -- of the write transaction.
  S_AXI_BRESP : out std_logic_vector(1 downto 0);
  -- Write response valid. This signal indicates that the channel
      -- is signaling a valid write response.
  S_AXI_BVALID  : out std_logic;
  -- Response ready. This signal indicates that the master
      -- can accept a write response.
  S_AXI_BREADY  : in std_logic;
  -- Read address (issued by master, acceped by Slave)
  S_AXI_ARADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  -- Protection type. This signal indicates the privilege
      -- and security level of the transaction, and whether the
      -- transaction is a data access or an instruction access.
```

```
    S_AXI_ARPROT  : in std_logic_vector(2 downto 0);
    -- Read address valid. This signal indicates that the channel
        -- is signaling valid read address and control information.
    S_AXI_ARVALID : in std_logic;
    -- Read address ready. This signal indicates that the slave is
        -- ready to accept an address and associated control signals.
    S_AXI_ARREADY : out std_logic;
    -- Read data (issued by slave)
    S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    -- Read response. This signal indicates the status of the
        -- read transfer.
    S_AXI_RRESP : out std_logic_vector(1 downto 0);
    -- Read valid. This signal indicates that the channel is
        -- signaling the required read data.
    S_AXI_RVALID  : out std_logic;
    -- Read ready. This signal indicates that the master can
        -- accept the read data and response information.
    S_AXI_RREADY  : in std_logic
  );
end blinker_v1_0_S00_AXI;

architecture arch_imp of blinker_v1_0_S00_AXI is

  -- AXI4LITE signals
  signal axi_awaddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_awready  : std_logic;
  signal axi_wready : std_logic;
  signal axi_bresp  : std_logic_vector(1 downto 0);
  signal axi_bvalid : std_logic;
  signal axi_araddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_arready  : std_logic;
  signal axi_rdata  : std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal axi_rresp  : std_logic_vector(1 downto 0);
  signal axi_rvalid : std_logic;

  -- Example-specific design signals
  -- local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
  -- ADDR_LSB is used for addressing 32/64 bit registers/memories
  -- ADDR_LSB = 2 for 32 bits (n downto 2)
  -- ADDR_LSB = 3 for 64 bits (n downto 3)
  constant ADDR_LSB  : integer := (C_S_AXI_DATA_WIDTH/32)+ 1;
  constant OPT_MEM_ADDR_BITS : integer := 1;
  ------------------------------------------------
  ---- Signals for user logic register space example
  ------------------------------------------------
  ---- Number of Slave Registers 4
  signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg_rden : std_logic;
  signal slv_reg_wren : std_logic;
  signal reg_data_out :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal byte_index : integer;
  signal aw_en  : std_logic;

begin
  -- I/O Connections assignments

  S_AXI_AWREADY <= axi_awready;
  S_AXI_WREADY  <= axi_wready;
  S_AXI_BRESP <= axi_bresp;
  S_AXI_BVALID  <= axi_bvalid;
```

```
S_AXI_ARREADY <= axi_arready;
S_AXI_RDATA <= axi_rdata;
S_AXI_RRESP <= axi_rresp;
S_AXI_RVALID  <= axi_rvalid;
-- Implement axi_awready generation
-- axi_awready is asserted for one S_AXI_ACLK clock cycle when both
-- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
-- de-asserted when reset is low.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_awready <= '0';
      aw_en <= '1';
    else
      if (axi_awready = '0' and S_AXI_AWVALID = '1' and S_AXI_WVALID = '1' and
          aw_en = '1') then
        -- slave is ready to accept write address when
        -- there is a valid write address and write data
        -- on the write address and data bus. This design
        -- expects no outstanding transactions.
          axi_awready <= '1';
          aw_en <= '0';
       elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then
          aw_en <= '1';
          axi_awready <= '0';
      else
        axi_awready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement axi_awaddr latching
-- This process is used to latch the address when both
-- S_AXI_AWVALID and S_AXI_WVALID are valid.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_awaddr <= (others => '0');
    else
      if (axi_awready = '0' and S_AXI_AWVALID = '1' and S_AXI_WVALID = '1' and
          aw_en = '1') then
        -- Write Address latching
        axi_awaddr <= S_AXI_AWADDR;
      end if;
    end if;
  end if;
end process;

-- Implement axi_wready generation
-- axi_wready is asserted for one S_AXI_ACLK clock cycle when both
-- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
-- de-asserted when reset is low.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_wready <= '0';
```

```
    else
      if (axi_wready = '0' and S_AXI_WVALID = '1' and S_AXI_AWVALID = '1' and
          aw_en = '1') then
          -- slave is ready to accept write data when
          -- there is a valid write address and write data
          -- on the write address and data bus. This design
          -- expects no outstanding transactions.
          axi_wready <= '1';
      else
        axi_wready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement memory mapped register select and write logic generation
-- The write data is accepted and written to memory mapped registers when
-- axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write
    strobes are used to
-- select byte enables of slave registers while writing.
-- These registers are cleared when reset (active low) is applied.
-- Slave register write enable is asserted when valid address and data are
    available
-- and the slave is ready to accept the write address and write data.
slv_reg_wren <= axi_wready and S_AXI_WVALID and axi_awready and S_AXI_AWVALID ;

process (S_AXI_ACLK)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
      slv_reg2 <= (others => '0');
      slv_reg3 <= (others => '0');
    else
      loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
      if (slv_reg_wren = '1') then
        case loc_addr is
          when b"00" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 0
                slv_reg0(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"01" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 1
                slv_reg1(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"10" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 2
```

```
                slv_reg2(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"11" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 3
                slv_reg3(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when others =>
            slv_reg0 <= slv_reg0;
            slv_reg1 <= slv_reg1;
            slv_reg2 <= slv_reg2;
            slv_reg3 <= slv_reg3;
        end case;
      end if;
    end if;
  end if;
end process;

-- Implement write response logic generation
-- The write response and response valid signals are asserted by the slave
-- when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
-- This marks the acceptance of address and indicates the status of
-- write transaction.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_bvalid  <= '0';
      axi_bresp   <= "00"; --need to work more on the responses
    else
      if (axi_awready = '1' and S_AXI_AWVALID = '1' and axi_wready = '1' and
          S_AXI_WVALID = '1' and axi_bvalid = '0'  ) then
        axi_bvalid <= '1';
        axi_bresp  <= "00";
      elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then   --check if bready
          is asserted while bvalid is high)
        axi_bvalid <= '0';                                 -- (there is a
            possibility that bready is always asserted high)
      end if;
    end if;
  end if;
end process;

-- Implement axi_arready generation
-- axi_arready is asserted for one S_AXI_ACLK clock cycle when
-- S_AXI_ARVALID is asserted. axi_awready is
-- de-asserted when reset (active low) is asserted.
-- The read address is also latched when S_AXI_ARVALID is
-- asserted. axi_araddr is reset to zero on reset assertion.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_arready <= '0';
      axi_araddr  <= (others => '1');
```

```
    else
      if (axi_aready = '0' and S_AXI_ARVALID = '1') then
        -- indicates that the slave has acceped the valid read address
        axi_aready <= '1';
        -- Read Address latching
        axi_araddr  <= S_AXI_ARADDR;
      else
        axi_aready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement axi_arvalid generation
-- axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
-- S_AXI_ARVALID and axi_aready are asserted. The slave registers
-- data are available on the axi_rdata bus at this instance. The
-- assertion of axi_rvalid marks the validity of read data on the
-- bus and axi_rresp indicates the status of read transaction.axi_rvalid
-- is deasserted on reset (active low). axi_rresp and axi_rdata are
-- cleared to zero on reset (active low).
process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_rvalid <= '0';
      axi_rresp  <= "00";
    else
      if (axi_aready = '1' and S_AXI_ARVALID = '1' and axi_rvalid = '0') then
        -- Valid read data is available at the read data bus
        axi_rvalid <= '1';
        axi_rresp  <= "00"; -- 'OKAY' response
      elsif (axi_rvalid = '1' and S_AXI_RREADY = '1') then
        -- Read data is accepted by the master
        axi_rvalid <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement memory mapped register select and read logic generation
-- Slave register read enable is asserted when valid address is available
-- and the slave is ready to accept the read address.
slv_reg_rden <= axi_aready and S_AXI_ARVALID and (not axi_rvalid) ;

process (slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr, S_AXI_ARESETN,
    slv_reg_rden)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
      when b"00" =>
        reg_data_out <= slv_reg0;
      when b"01" =>
        reg_data_out <= slv_reg1;
      when b"10" =>
        reg_data_out <= slv_reg2;
      when b"11" =>
        reg_data_out <= slv_reg3;
      when others =>
        reg_data_out  <= (others => '0');
    end case;
```

```
  end process;

  -- Output register or memory read data
  process( S_AXI_ACLK ) is
  begin
    if (rising_edge (S_AXI_ACLK)) then
      if ( S_AXI_ARESETN = '0' ) then
        axi_rdata  <= (others => '0');
      else
        if (slv_reg_rden = '1') then
          -- When there is a valid read address (S_AXI_ARVALID) with
          -- acceptance of read address by the slave (axi_aready),
          -- output the read dada
          -- Read address mux
            axi_rdata <= reg_data_out;     -- register read data
        end if;
      end if;
    end if;
  end process;


  -- Add user logic here
  --added to read value of the first 8 bits of the first register
    OUTPUT_LED <= slv_reg0 (7 downto 0);
  -- User logic ends

end arch_imp;
```

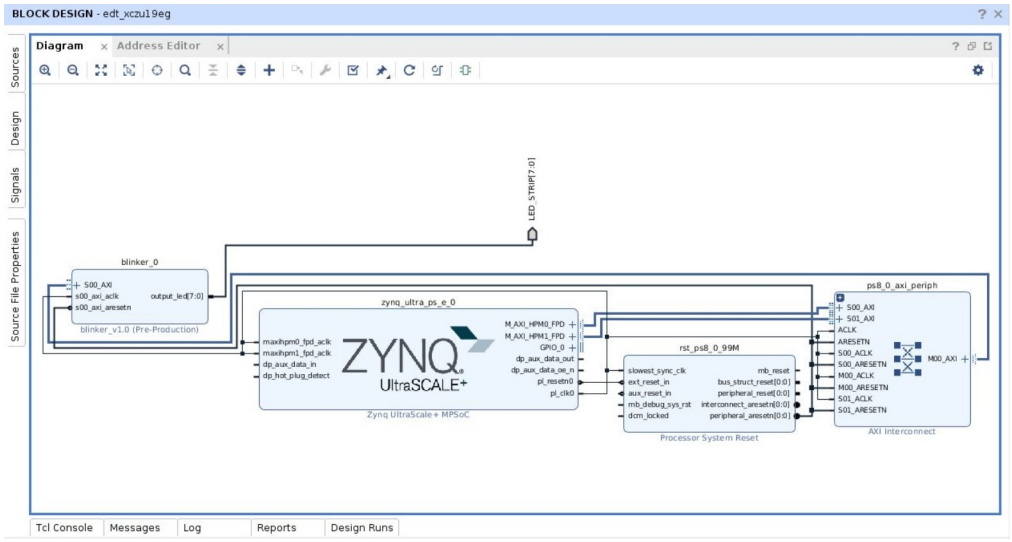

Figure C.2. Block diagram of the test_up_down design.

Listing C.8. helloworld.c C program

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "stdlib.h"



int main()
{
    init_platform();

    volatile unsigned int* blinker = (volatile unsigned int*) 0xa0000000;

        int strip = 0b00000001;
        int cont = 0;
        int delayer = 0;
        int up = 1;

        while(cont<80){

          if(up==1){
            strip = strip << 1;
          }else{
            strip = strip >>1;
          }

          if (strip == 0b10000000){
            up=0;
          }

          if (strip == 0b00000001){
            up=1;
          }



          blinker[0]=strip;
          while(delayer<10001000){
            delayer++;
          }
          delayer = 0;
          cont++;
      }

    print("Hello World\n\r");

    cleanup_platform();
    return 0;
}
```

C.4 Test of processor interrupts

Listing C.9. mb_1_TA1.xdc constraints file

```
# FPGA module FM-XCZU19EG-R2
```

```
# pins which are not connected to an x-board


set_property PACKAGE_PIN A23 [get_ports LED_BLUE]
set_property PACKAGE_PIN J20 [get_ports LED_BLUE2]
##############################################################
# UD001-3.19-Hardware-UserManual.docx p. 31, 120, 122
set_property IOSTANDARD LVCMOS18 [get_ports LED_BLUE]
set_property IOSTANDARD LVCMOS18 [get_ports LED_BLUE2]
##############################################################
set_property PACKAGE_PIN F27 [get_ports LED_GREEN]
set_property IOSTANDARD LVCMOS18 [get_ports LED_GREEN]
set_property PACKAGE_PIN E20 [get_ports LED_GREEN2]
set_property IOSTANDARD LVCMOS18 [get_ports LED_GREEN2]
set_property PACKAGE_PIN C26 [get_ports LED_RED]
set_property IOSTANDARD LVCMOS18 [get_ports LED_RED]
set_property PACKAGE_PIN D20 [get_ports LED_RED2]
set_property IOSTANDARD LVCMOS18 [get_ports LED_RED2]
set_property PACKAGE_PIN A22 [get_ports LED_YELLOW]
set_property IOSTANDARD LVCMOS18 [get_ports LED_YELLOW]
set_property PACKAGE_PIN M28 [get_ports LED_YELLOW2]
set_property IOSTANDARD LVCMOS18 [get_ports LED_YELLOW2]


# pins which are connected to motherboard connector TB1
# and connector BA1 on x-board EB-PDS-DEBUG-R1
##############################################################

# UD001-3.19-Hardware-UserManual.docx p. 214

##############################################################

set_property PACKAGE_PIN U39 [get_ports tb1_eb1_IO_000]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_000]
set_property PACKAGE_PIN V39 [get_ports tb1_eb1_IO_001]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_001]
set_property PACKAGE_PIN R38 [get_ports tb1_eb1_IO_002]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_002]
set_property PACKAGE_PIN T37 [get_ports tb1_eb1_IO_003]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_003]
set_property PACKAGE_PIN N39 [get_ports tb1_eb1_IO_004]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_004]
set_property PACKAGE_PIN N38 [get_ports tb1_eb1_IO_005]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_005]
set_property PACKAGE_PIN P37 [get_ports tb1_eb1_IO_006]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_006]
set_property PACKAGE_PIN R37 [get_ports tb1_eb1_IO_007]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_007]
set_property PACKAGE_PIN P39 [get_ports tb1_eb1_IO_008]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_008]
set_property PACKAGE_PIN R39 [get_ports tb1_eb1_IO_009]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_009]
set_property PACKAGE_PIN T38 [get_ports tb1_eb1_IO_010]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_010]
set_property PACKAGE_PIN U38 [get_ports tb1_eb1_IO_011]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_011]
set_property PACKAGE_PIN V27 [get_ports tb1_eb1_IO_012]
set_property PACKAGE_PIN V26 [get_ports tb1_eb1_IO_013]
set_property PACKAGE_PIN U29 [get_ports tb1_eb1_IO_014]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_014]
set_property PACKAGE_PIN U28 [get_ports tb1_eb1_IO_015]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_015]
```

```
# VREF/DCI constraints for banks related to connector BB2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 72]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 73]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 74]

# VREF/DCI constraints for banks related to connector TA2
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 66]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 64]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 65]

# VREF/DCI constraints for banks related to connector TB1
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 67]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 69]
#   #set_property INTERNAL_VREF {0.90} [get_iobanks 68]

set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_012]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_IO_013]
```

Listing C.10. TOP.vhd TOP file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Library UNISIM;
use UNISIM.vcomponents.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TOP is
port (
   LED_BLUE : out std_logic;
   LED_BLUE2 : out std_logic;
   LED_GREEN : out std_logic;
   LED_GREEN2 : out std_logic;
   LED_RED : out std_logic;
   LED_RED2 : out std_logic;
   LED_YELLOW : out std_logic;
   LED_YELLOW2 : out std_logic;
   -- -- pins which are connected to motherboard connector TB1
   -- -- and connector BA1 on x-board EB-PDS-DEBUG-R1

   -- ##### ud001 pag 212 fig 150 #### questi sono i 37 gpio pin header(da 000 a
       015 sono connessi anche i led)
   tb1_eb1_IO_000: inout std_logic;
   tb1_eb1_IO_001: inout std_logic;
   tb1_eb1_IO_002: inout std_logic;
   tb1_eb1_IO_003: inout std_logic;
   tb1_eb1_IO_004: inout std_logic;
   tb1_eb1_IO_005: inout std_logic;
   tb1_eb1_IO_006: inout std_logic;
   tb1_eb1_IO_007: inout std_logic;
   tb1_eb1_IO_008: inout std_logic;
   tb1_eb1_IO_009: inout std_logic;
```

```
    tb1_eb1_IO_010: inout std_logic;
    tb1_eb1_IO_011: inout std_logic;
    tb1_eb1_IO_012: inout std_logic;
    tb1_eb1_IO_013: inout std_logic;
    tb1_eb1_IO_014: inout std_logic;
    tb1_eb1_IO_015: inout std_logic

    );
end TOP;

architecture Behavioral of TOP is

component edt_xczu19eg_wrapper is
  port (
    gpio_rtl_0_tri_o : out STD_LOGIC_VECTOR ( 15 downto 0 );
    gpio_rtl_1_tri_o : out STD_LOGIC_VECTOR ( 7 downto 0 )
  );
end component;

signal led_debug : std_logic_vector (15 downto 0);
signal led_board  : std_logic_vector (7 downto 0);

begin

processore : edt_xczu19eg_wrapper port map (gpio_rtl_0_tri_o => led_debug,
                                            gpio_rtl_1_tri_o => led_board);

    tb1_eb1_IO_000 <= led_debug(0);
    tb1_eb1_IO_001 <= led_debug(1);
    tb1_eb1_IO_002 <= led_debug(2);
    tb1_eb1_IO_003 <= led_debug(3);
    tb1_eb1_IO_004 <= led_debug(4);
    tb1_eb1_IO_005 <= led_debug(5);
    tb1_eb1_IO_006 <= led_debug(6);
    tb1_eb1_IO_007 <= led_debug(7);
    tb1_eb1_IO_008 <= led_debug(8);
    tb1_eb1_IO_009 <= led_debug(9);
    tb1_eb1_IO_010 <= led_debug(10);
    tb1_eb1_IO_011 <= led_debug(11);
    tb1_eb1_IO_012 <= led_debug(12);
    tb1_eb1_IO_013 <= led_debug(13);
    tb1_eb1_IO_014 <= led_debug(14);
    tb1_eb1_IO_015 <= led_debug(15);


    LED_GREEN <= led_board(0);
    LED_RED <= led_board(1);
    LED_YELLOW <= led_board(2);
    LED_BLUE <= led_board(3);
    LED_GREEN2 <= led_board(4);
    LED_RED2 <= led_board(5);
    LED_YELLOW2 <= led_board(6);
    LED_BLUE2 <= led_board(7);




end Behavioral;
```

Listing C.11. helloworld.c C program

```
#include <stdio.h>
```

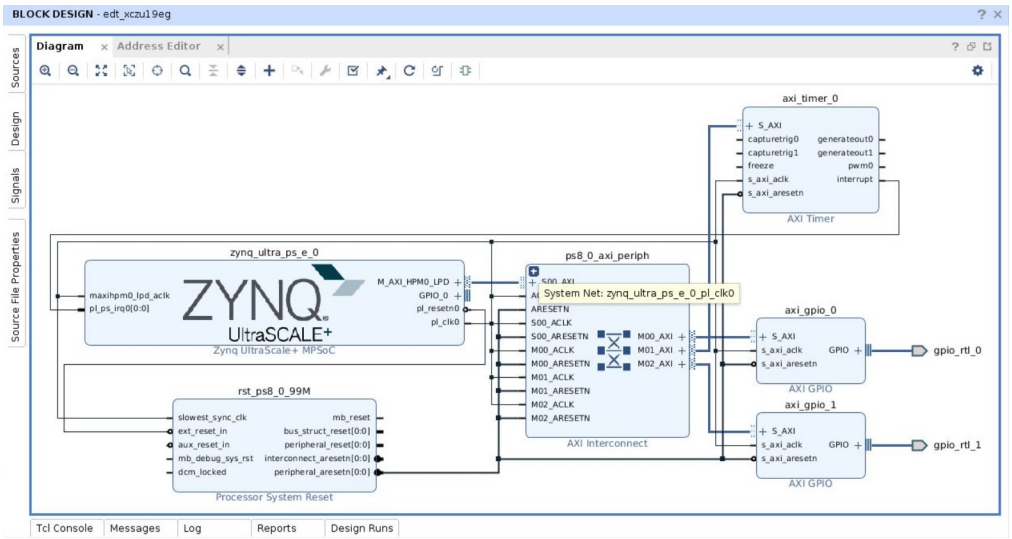

Figure C.3. Block diagram of the test_interrupt design.

```
#include "platform.h"
#include "xil_printf.h"


// gpio
#include <xgpio.h>
#include "xparameters.h"
#include "sleep.h"

//interrupt
#include "xtmrctr.h"
#include "xscugic.h"


//variables  gpio
  XGpio dbg,zynq;

//variables interrupt
  XScuGic intCtrl;
  XTmrCtr timer;

int turn=0,times=0;

//handler dell'interrupt in cui fare le operazioni

void timerInterruptHandler(void *userParam , u8 TmrCtrNumber){
  if(times==4){
    if(turn==0){

          XGpio_DiscreteWrite(&zynq,1,0b00001111);
          XGpio_DiscreteWrite(&dbg,1,0b0000111111110000);
          turn=1;
        }else{

          XGpio_DiscreteWrite(&zynq,1,0b11110000);
          XGpio_DiscreteWrite(&dbg,1,0b1111000000001111);
          turn=0;
```

```
        }
    times=0;
  }
  times++;
}


int main()
{


  int cont=0;

  //initialize gpio
  XGpio_Initialize(&dbg,XPAR_AXI_GPIO_0_DEVICE_ID);
  XGpio_Initialize(&zynq,XPAR_AXI_GPIO_1_DEVICE_ID);

  //setting direction
  XGpio_SetDataDirection(&dbg,1,0x0);
  XGpio_SetDataDirection(&zynq,1,0x0);

  //initialize interrupt
  int xStatus = XTmrCtr_Initialize(&timer,XPAR_AXI_TIMER_0_DEVICE_ID);
  if (xStatus != XST_SUCCESS){

    printf("Could not initialize timer\r\n");
  }

  XTmrCtr_SetHandler(&timer,(XTmrCtr_Handler)timerInterruptHandler,&timer);

  //initialize the interrupt controller driver so that it is ready to use
  XScuGic_Config *IntcConfig = XScuGic_LookupConfig(XPAR_SCUGIC_SINGLE_DEVICE_ID);

  int Status =
      XScuGic_CfgInitialize(&intCtrl,IntcConfig,IntcConfig->CpuBaseAddress);

  XScuGic_SetPriorityTriggerType(&intCtrl,XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR,
      0xA0,0x3);
  Status = XScuGic_Connect(&intCtrl,XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR,
              (Xil_InterruptHandler)XTmrCtr_InterruptHandler,&timer);

  XScuGic_Enable(&intCtrl,XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR);

  //initialize the exception table
  Xil_ExceptionInit();

  //register the interrupt controller handler with the exception table
  Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                (Xil_ExceptionHandler)XScuGic_InterruptHandler,
                (void *) &intCtrl);

  //enable exceptions
  Xil_ExceptionEnable();

  XTmrCtr_SetOptions(&timer,0,XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);
  XTmrCtr_SetResetValue(&timer,0,0xFFE17B80); //50 hz
  XTmrCtr_Start(&timer,0);


  init_platform();


    while(cont<100){
```

```
        cont++;
    }


    cleanup_platform();
    return 0;
}
```

C.5 Test of Gbit Ethernet

Listing C.12. mb_1_TA1.xdc constraints file

```
# pins which are connected to motherboard connector TB2
# and connector BA1 on x-board EB-PDS-GBITETHERNET-R1
set_property  PACKAGE_PIN AF13 [get_ports {tb2_eb1_ETH1_CLK_TO_MAC}]
set_property  PACKAGE_PIN N13 [get_ports {tb2_eb1_ETH1_COL_CLK_MAC_FREQ}]
set_property  PACKAGE_PIN N12 [get_ports {tb2_eb1_ETH1_CRS_RGMII_SEL0}]
set_property  PACKAGE_PIN M12 [get_ports {tb2_eb1_ETH1_GTX_CLK_TCK}]
set_property  PACKAGE_PIN G14 [get_ports {tb2_eb1_ETH1_MDC}]
set_property  PACKAGE_PIN H14 [get_ports {tb2_eb1_ETH1_MDIO}]
set_property  PACKAGE_PIN U14 [get_ports {tb2_eb1_ETH1_NINTERRUPT}]
set_property  PACKAGE_PIN W12 [get_ports {tb2_eb1_ETH1_NRESET}]
set_property  PACKAGE_PIN V13 [get_ports {tb2_eb1_ETH1_RX_CLK}]
set_property  PACKAGE_PIN V14 [get_ports {tb2_eb1_ETH1_RX_DV_RCK}]
set_property  PACKAGE_PIN M13 [get_ports {tb2_eb1_ETH1_RX_ER_RXDV_ER}]
set_property  PACKAGE_PIN N16 [get_ports {tb2_eb1_ETH1_RXD0_RX0}]
set_property  PACKAGE_PIN P16 [get_ports {tb2_eb1_ETH1_RXD1_RX1}]
set_property  PACKAGE_PIN L16 [get_ports {tb2_eb1_ETH1_RXD2_RX2}]
set_property  PACKAGE_PIN M16 [get_ports {tb2_eb1_ETH1_RXD3_RX3}]
set_property  PACKAGE_PIN AG13 [get_ports {tb2_eb1_ETH1_TX_CLK_RGMII_SEL1}]
set_property  PACKAGE_PIN J15 [get_ports {tb2_eb1_ETH1_TX_EN_TXEN_ER}]
set_property  PACKAGE_PIN K15 [get_ports {tb2_eb1_ETH1_TX_ER}]
set_property  PACKAGE_PIN L12 [get_ports {tb2_eb1_ETH1_TXD0_TX0}]
set_property  PACKAGE_PIN L13 [get_ports {tb2_eb1_ETH1_TXD1_TX1}]
set_property  PACKAGE_PIN K12 [get_ports {tb2_eb1_ETH1_TXD2_TX2}]
set_property  PACKAGE_PIN K13 [get_ports {tb2_eb1_ETH1_TXD3_TX3}]
set_property  PACKAGE_PIN J14 [get_ports {tb2_eb1_ETH1_TXD4}]
set_property  PACKAGE_PIN K14 [get_ports {tb2_eb1_ETH1_TXD5}]
set_property  PACKAGE_PIN G13 [get_ports {tb2_eb1_ETH1_TXD6}]
set_property  PACKAGE_PIN H13 [get_ports {tb2_eb1_ETH1_TXD7}]

set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_GTX_CLK_TCK]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RX_DV_RCK]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RX_ER_RXDV_ER]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RXD0_RX0]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RXD1_RX1]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RXD2_RX2]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RXD3_RX3]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TX_EN_TXEN_ER]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD0_TX0]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD1_TX1]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD2_TX2]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD3_TX3]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_MDC]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_MDIO]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_CLK_TO_MAC]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_COL_CLK_MAC_FREQ]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_CRS_RGMII_SEL0]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_NINTERRUPT]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_NRESET]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_RX_CLK]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TX_CLK_RGMII_SEL1]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TX_ER]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD4]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD5]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD6]
set_property IOSTANDARD LVCMOS18 [get_ports tb1_eb1_ETH1_TXD7]


# VREF/DCI constraints for banks related to connector BB2
#   set_property INTERNAL_VREF {0.90} [get_iobanks 72]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 73]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 74]

# VREF/DCI constraints for banks related to connector TA2
#   set_property INTERNAL_VREF {0.90} [get_iobanks 66]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 64]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 65]

# VREF/DCI constraints for banks related to connector TB1
#   set_property INTERNAL_VREF {0.90} [get_iobanks 67]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 69]
#   set_property INTERNAL_VREF {0.90} [get_iobanks 68]
```

Listing C.13. mb_1_TA1.vhd TOP file

```
-- FPGA module FM-XCZU19EG-R2

library ieee;
use ieee.std_logic_1164.all;
Library UNISIM;
use UNISIM.vcomponents.all;

entity mb_1_TA1 is
  port (

   -- pins which are connected to motherboard connector TA2
   -- and connector BA1 on x-board EB-PDS-GBITETHERNET-R1
   tb1_eb1_ETH1_CLK_TO_MAC: in std_logic;
   tb1_eb1_ETH1_COL_CLK_MAC_FREQ: in std_logic;
   tb1_eb1_ETH1_CRS_RGMII_SEL0: in std_logic;
   tb1_eb1_ETH1_GTX_CLK_TCK: out std_logic;
   tb1_eb1_ETH1_MDC: out std_logic;
   tb1_eb1_ETH1_MDIO: inout std_logic;
   tb1_eb1_ETH1_NINTERRUPT: in std_logic;
   tb1_eb1_ETH1_NRESET: out std_logic;
   tb1_eb1_ETH1_RX_CLK: in std_logic;
   tb1_eb1_ETH1_RX_DV_RCK: in std_logic;
   tb1_eb1_ETH1_RX_ER_RXDV_ER: in std_logic;
   tb1_eb1_ETH1_RXD0_RX0: in std_logic;
   tb1_eb1_ETH1_RXD1_RX1: in std_logic;
   tb1_eb1_ETH1_RXD2_RX2: in std_logic;
   tb1_eb1_ETH1_RXD3_RX3: in std_logic;
   tb1_eb1_ETH1_TX_CLK_RGMII_SEL1: in std_logic;
   tb1_eb1_ETH1_TX_EN_TXEN_ER: out std_logic;
   tb1_eb1_ETH1_TX_ER: out std_logic;
   tb1_eb1_ETH1_TXD0_TX0: out std_logic;
   tb1_eb1_ETH1_TXD1_TX1: out std_logic;
   tb1_eb1_ETH1_TXD2_TX2: out std_logic;
```

```
    tb1_eb1_ETH1_TXD3_TX3: out std_logic;
    tb1_eb1_ETH1_TXD4: out std_logic;
    tb1_eb1_ETH1_TXD5: out std_logic;
    tb1_eb1_ETH1_TXD6: out std_logic;
    tb1_eb1_ETH1_TXD7: out std_logic

  );
end entity mb_1_TA1;

architecture STRUCTURE of mb_1_TA1 is

signal MDIO_PHY_0_mdio_i, MDIO_PHY_0_mdio_o, MDIO_PHY_0_mdio_t : STD_LOGIC;

  component design_1 is
  port (

    MDIO_PHY_0_mdc : out STD_LOGIC;
    MDIO_PHY_0_mdio_i : in STD_LOGIC;
    MDIO_PHY_0_mdio_o : out STD_LOGIC;
    MDIO_PHY_0_mdio_t : out STD_LOGIC;
    RGMII_0_rd : in STD_LOGIC_VECTOR ( 3 downto 0 );
    RGMII_0_rx_ctl : in STD_LOGIC;
    RGMII_0_rxc : in STD_LOGIC;
    RGMII_0_td : out STD_LOGIC_VECTOR ( 3 downto 0 );
    RGMII_0_tx_ctl : out STD_LOGIC;
    RGMII_0_txc : out STD_LOGIC;
    PHY_NRESET : out STD_LOGIC

  );
  end component design_1;


begin

design_1_i: component design_1
     port map (

       MDIO_PHY_0_mdc => tb1_eb1_ETH1_MDC,
       MDIO_PHY_0_mdio_i => MDIO_PHY_0_mdio_i,
       MDIO_PHY_0_mdio_o => MDIO_PHY_0_mdio_o,
       MDIO_PHY_0_mdio_t => MDIO_PHY_0_mdio_t,
       RGMII_0_rd(0) => tb1_eb1_ETH1_RXD0_RX0,
       RGMII_0_rd(1) => tb1_eb1_ETH1_RXD1_RX1,
       RGMII_0_rd(2) => tb1_eb1_ETH1_RXD2_RX2,
       RGMII_0_rd(3) => tb1_eb1_ETH1_RXD3_RX3,
       RGMII_0_rx_ctl => tb1_eb1_ETH1_RX_ER_RXDV_ER,
       RGMII_0_rxc => tb1_eb1_ETH1_RX_DV_RCK,
       RGMII_0_td(0) => tb1_eb1_ETH1_TXD0_TX0,
       RGMII_0_td(1) => tb1_eb1_ETH1_TXD1_TX1,
       RGMII_0_td(2) => tb1_eb1_ETH1_TXD2_TX2,
       RGMII_0_td(3) => tb1_eb1_ETH1_TXD3_TX3,
       RGMII_0_tx_ctl => tb1_eb1_ETH1_TX_EN_TXEN_ER,
       RGMII_0_txc => tb1_eb1_ETH1_GTX_CLK_TCK,
       PHY_NRESET => tb1_eb1_ETH1_NRESET,


     );


--GBE
--tc2_eb1_ETH1_COL_CLK_MAC_FREQ <= '1'; --CLK TO MAC 125MHZ (unused)
tb1_eb1_ETH1_TX_ER <= '1'; --unused in RGMII
tb1_eb1_ETH1_TXD4<= '0';
```

```
tb1_eb1_ETH1_TXD5<= '0';
tb1_eb1_ETH1_TXD6<= '0';
tb1_eb1_ETH1_TXD7 <= '0';

end STRUCTURE;
```

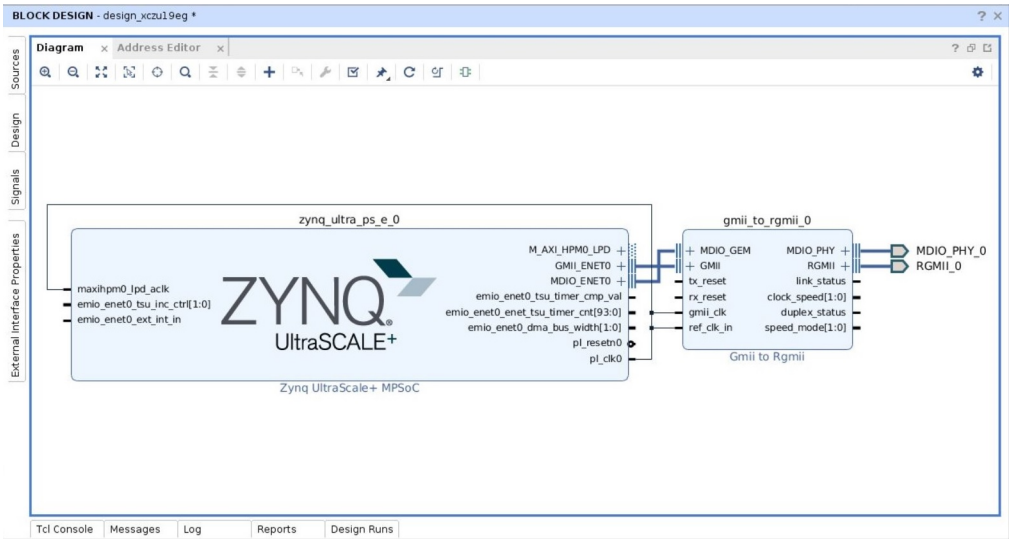

Figure C.4. Block diagram of the test_client design.

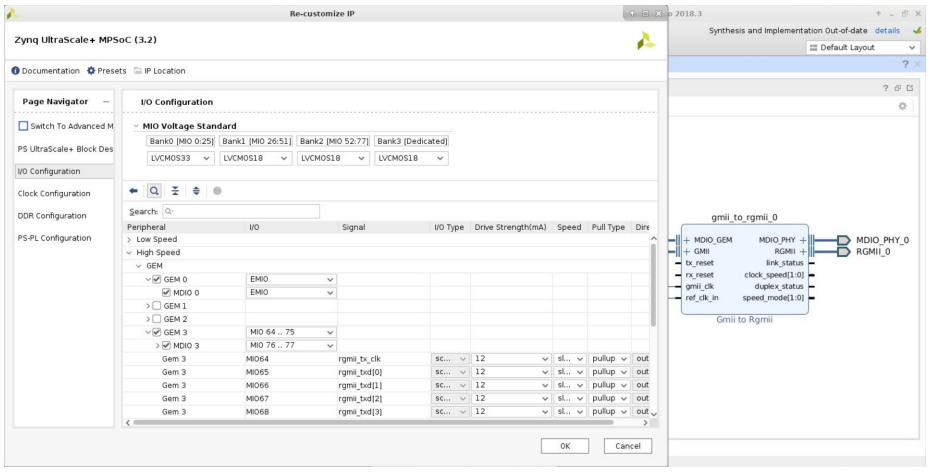

Figure C.5. Zynq Ip settings for the Ethernet port pins.

Listing C.14. client.c C program

```
#include <stdio.h>
#include <stdlib.h>
```

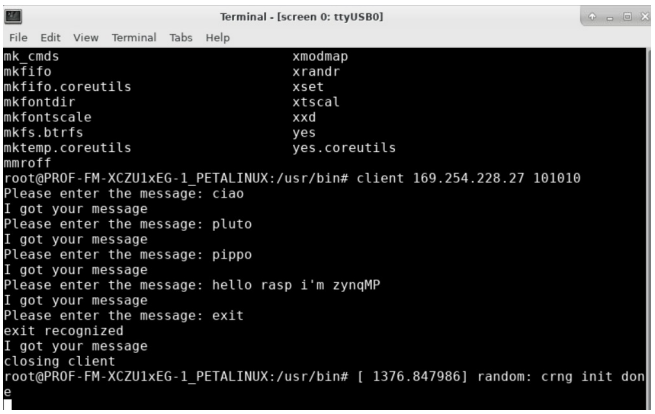

Figure C.6. Test of the Ethernet with client running on the Zynq.

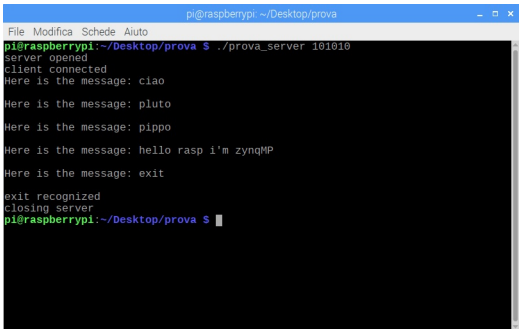

Figure C.7. Test of the Ethernet with server running on the Raspberry.

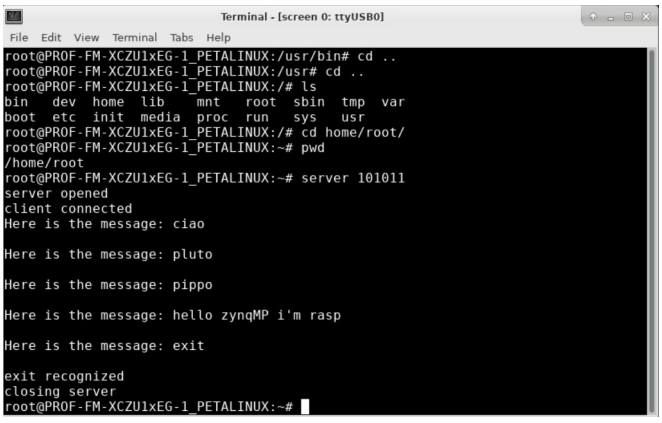

Figure C.8. Test of the Ethernet with server running on the Zynq.

```
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
```

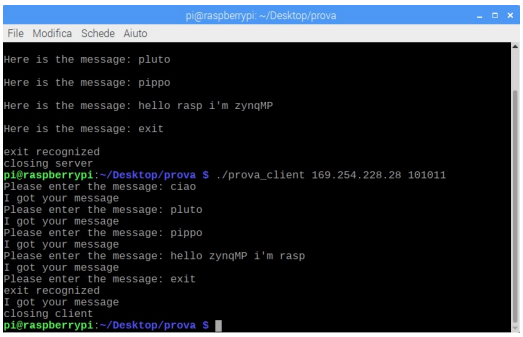

Figure C.9. Test of the Ethernet with client running on the Raspberry.

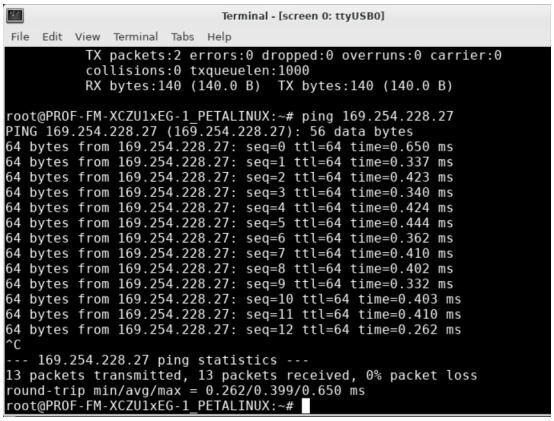

Figure C.10. Ping test from Zynq to raspberry.

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n,out=0;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if (sockfd < 0)
         error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
         fprintf(stderr,"ERROR, no such host\n");
         exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
          (char *)&serv_addr.sin_addr.s_addr,
          server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
         error("ERROR connecting");



    while(out!=1){

         printf("Please enter the message: ");
         bzero(buffer,256);
         fgets(buffer,255,stdin);
         n = write(sockfd,buffer,strlen(buffer));
         if (n < 0)
              error("ERROR writing to socket");

         if(strncmp(buffer,"exit",4)==0){
              printf("exit recognized\n");
              out=1;
         }

         bzero(buffer,256);

         n = read(sockfd,buffer,255);
         if (n < 0)
              error("ERROR reading from socket");
         printf("%s\n",buffer);



    }

    printf("closing client\n");
    close(sockfd);
    return 0;
}
```

Listing C.15. server.c C program

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(const char *msg)
{
```

```
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno,out=0;
     socklen_t clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     printf("server opened\n");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd,
                 (struct sockaddr *) &cli_addr,
                 &clilen);
     if (newsockfd < 0)
          error("ERROR on accept");
     printf("client connected\n");

     while(out!=1){
          bzero(buffer,256);
          n = read(newsockfd,buffer,255);
          if (n < 0) error("ERROR reading from socket");
          printf("Here is the message: %s\n",buffer);
          if(strncmp(buffer,"exit",4)==0){
             printf("exit recognized\n");
             out=1;
        }
          n = write(newsockfd,"I got your message",18);
          if (n < 0) error("ERROR writing to socket");

          }
     printf("closing server\n");
     close(newsockfd);
     close(sockfd);

     return 0;
}
```

C.6 Test of DDR4

Listing C.16. TOP.vhd TOP file

```
-- FPGA module FM-XCZU19EG-R2

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Library UNISIM;
use UNISIM.vcomponents.all;

entity mb_1_TA1 is
  port (

    LED_BLUE : out std_logic;
    LED_BLUE2 : out std_logic;
    LED_GREEN : out std_logic;
    LED_GREEN2 : out std_logic;
    LED_RED : out std_logic;
    LED_RED2 : out std_logic;
    LED_YELLOW : out std_logic;
    LED_YELLOW2 : out std_logic;


   -- -- pins which are connected to motherboard connector BB2
   -- -- and connector BA1 on x-board EB-PDS-DDR4-R6
   bb2_eb1_CLK_IN_N: in std_logic; --
   bb2_eb1_CLK_IN_P: in std_logic; --
   bb2_eb1_DDR4_A0          : out std_logic;--
   bb2_eb1_DDR4_A1          : out std_logic;--
   bb2_eb1_DDR4_A2          : out std_logic;--
   bb2_eb1_DDR4_A3          : out std_logic;--
   bb2_eb1_DDR4_A4          : out std_logic;--
   bb2_eb1_DDR4_A5          : out std_logic;--
   bb2_eb1_DDR4_A6          : out std_logic;--
   bb2_eb1_DDR4_A7          : out std_logic;--
   bb2_eb1_DDR4_A8          : out std_logic;--
   bb2_eb1_DDR4_A9          : out std_logic;--
   bb2_eb1_DDR4_A10_AP      : out std_logic;--
   bb2_eb1_DDR4_A11         : out std_logic;--
   bb2_eb1_DDR4_A12_BC_N    : out std_logic;--
   bb2_eb1_DDR4_A13         : out std_logic;--
   bb2_eb1_DDR4_A14_WE_N    : out std_logic;--
   bb2_eb1_DDR4_A15_CAS_N   : out std_logic;--
   bb2_eb1_DDR4_A16_RAS_N   : out std_logic;--
   bb2_eb1_DDR4_ACT_N       : out std_logic;--
   bb2_eb1_DDR4_ALERT_N     : in std_logic; --
   bb2_eb1_DDR4_BA0         : out std_logic;--
   bb2_eb1_DDR4_BA1         : out std_logic;--
   bb2_eb1_DDR4_BG0         : out std_logic;--
   bb2_eb1_DDR4_CK_C        : out std_logic;--
   bb2_eb1_DDR4_CK_T        : out std_logic;--
   bb2_eb1_DDR4_CKE         : out std_logic;--
   bb2_eb1_DDR4_CS_N        : out std_logic;--
   bb2_eb1_DDR4_DQ00_D0_DX00  : inout std_logic;--
   bb2_eb1_DDR4_DQ00_D1_DX01  : inout std_logic;--
   bb2_eb1_DDR4_DQ00_D2_DX02  : inout std_logic;--
   bb2_eb1_DDR4_DQ00_D3_DX03  : inout std_logic;--
   bb2_eb1_DDR4_DQ00_D4_DX04  : inout std_logic;--
```

```
bb2_eb1_DDR4_DQ00_D5_DX05   : inout std_logic;--
bb2_eb1_DDR4_DQ00_D6_DX06   : inout std_logic;--
bb2_eb1_DDR4_DQ00_D7_DX07   : inout std_logic;--
bb2_eb1_DDR4_DQ00_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ00_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ00_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ01_D0_DX08   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D1_DX09   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D2_DX10   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D3_DX11   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D4_DX12   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D5_DX13   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D6_DX14   : inout std_logic;--
bb2_eb1_DDR4_DQ01_D7_DX15   : inout std_logic;--
bb2_eb1_DDR4_DQ01_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ01_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ01_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ02_D0_DX16   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D1_DX17   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D2_DX18   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D3_DX19   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D4_DX20   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D5_DX21   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D6_DX22   : inout std_logic;--
bb2_eb1_DDR4_DQ02_D7_DX23   : inout std_logic;--
bb2_eb1_DDR4_DQ02_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ02_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ02_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ03_D0_DX24   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D1_DX25   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D2_DX26   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D3_DX27   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D4_DX28   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D5_DX29   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D6_DX30   : inout std_logic;--
bb2_eb1_DDR4_DQ03_D7_DX31   : inout std_logic;--
bb2_eb1_DDR4_DQ03_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ03_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ03_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ04_D0_DX32   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D1_DX33   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D2_DX34   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D3_DX35   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D4_DX36   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D5_DX37   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D6_DX38   : inout std_logic;--
bb2_eb1_DDR4_DQ04_D7_DX39   : inout std_logic;--
bb2_eb1_DDR4_DQ04_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ04_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ04_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ05_D0_DX40   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D1_DX41   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D2_DX42   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D3_DX43   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D4_DX44   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D5_DX45   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D6_DX46   : inout std_logic;--
bb2_eb1_DDR4_DQ05_D7_DX47   : inout std_logic;--
bb2_eb1_DDR4_DQ05_DM_DBI_N  : inout std_logic;--
bb2_eb1_DDR4_DQ05_DQS_C     : inout std_logic;--
bb2_eb1_DDR4_DQ05_DQS_T     : inout std_logic;--
bb2_eb1_DDR4_DQ06_D0_DX48   : inout std_logic;--
bb2_eb1_DDR4_DQ06_D1_DX49   : inout std_logic;--
```

```
    bb2_eb1_DDR4_DQ06_D2_DX50   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_D3_DX51   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_D4_DX52   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_D5_DX53   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_D6_DX54   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_D7_DX55   : inout std_logic;--
    bb2_eb1_DDR4_DQ06_DM_DBI_N  : inout std_logic;--
    bb2_eb1_DDR4_DQ06_DQS_C     : inout std_logic;--
    bb2_eb1_DDR4_DQ06_DQS_T     : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D0_DX56   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D1_DX57   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D2_DX58   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D3_DX59   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D4_DX60   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D5_DX61   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D6_DX62   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_D7_DX63   : inout std_logic;--
    bb2_eb1_DDR4_DQ07_DM_DBI_N  : inout std_logic;--
    bb2_eb1_DDR4_DQ07_DQS_C     : inout std_logic;--
    bb2_eb1_DDR4_DQ07_DQS_T     : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D0_DX64   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D1_DX65   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D2_DX66   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D3_DX67   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D4_DX68   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D5_DX69   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D6_DX70   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_D7_DX71   : inout std_logic;--
    bb2_eb1_DDR4_DQ08_DM_DBI_N  : inout std_logic;--
    bb2_eb1_DDR4_DQ08_DQS_C     : inout std_logic;--
    bb2_eb1_DDR4_DQ08_DQS_T     : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D0_DX72   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D1_DX73   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D2_DX74   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D3_DX75   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D4_DX76   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D5_DX77   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D6_DX78   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_D7_DX79   : inout std_logic;--
    bb2_eb1_DDR4_DQ09_DM_DBI_N  : inout std_logic;--
    bb2_eb1_DDR4_DQ09_DQS_C     : inout std_logic;--
    bb2_eb1_DDR4_DQ09_DQS_T     : inout std_logic;--
    bb2_eb1_DDR4_ODT            : out std_logic;--
    bb2_eb1_DDR4_PAR            : out std_logic;--
    bb2_eb1_DDR4_RESET_N: out std_logic;--
    bb2_eb1_DDR4_TEN: out std_logic;--
    bb2_eb1_LED_BLUE1     : out std_logic;--
    bb2_eb1_LED_BLUE2     : out std_logic;--
    bb2_eb1_LED_GREEN1     : out std_logic;--
    bb2_eb1_LED_GREEN2    : out std_logic;--
    bb2_eb1_LED_RED1      : out std_logic;--
    bb2_eb1_LED_RED2      : out std_logic;--
    bb2_eb1_LED_YELLOW1   : out std_logic;--
    bb2_eb1_LED_YELLOW2   : out std_logic--
  );
end entity mb_1_TA1;

architecture beh of mb_1_TA1 is

component edt_xczu19eg_wrapper is
  port (

    ddr4_rtl_0_act_n : out STD_LOGIC;
```

```
    ddr4_rtl_0_adr : out STD_LOGIC_VECTOR ( 16 downto 0 );
    ddr4_rtl_0_ba : out STD_LOGIC_VECTOR ( 1 downto 0 );
    ddr4_rtl_0_bg : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_ck_c : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_ck_t : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_cke : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_cs_n : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_dm_n : inout STD_LOGIC_VECTOR ( 7 downto 0 );
    ddr4_rtl_0_dq : inout STD_LOGIC_VECTOR ( 63 downto 0 );
    ddr4_rtl_0_dqs_c : inout STD_LOGIC_VECTOR ( 7 downto 0 );
    ddr4_rtl_0_dqs_t : inout STD_LOGIC_VECTOR ( 7 downto 0 );
    ddr4_rtl_0_odt : out STD_LOGIC_VECTOR ( 0 to 0 );
    ddr4_rtl_0_reset_n : out STD_LOGIC;
    diff_clock_rtl_0_clk_n : inout STD_LOGIC;
    diff_clock_rtl_0_clk_p : inout STD_LOGIC;
    reset_rtl_0 : in STD_LOGIC
  );
end component;


  signal clk_n,clk_p,reset_n,reset_rtl,act_n : std_logic;
  signal adr : std_logic_vector(16 downto 0);
  signal ba : std_logic_vector(1 downto 0);
  signal bg,ck_c,ck_t,cke,cs_n,odt : std_logic_vector(0 downto 0);
  signal dm_n,dqs_c,dqs_t : std_logic_vector(7 downto 0);
  signal dq : std_logic_vector(63 downto 0);

begin


system : edt_xczu19eg_wrapper port map (


  diff_clock_rtl_0_clk_n => clk_n,
    diff_clock_rtl_0_clk_p => clk_p,
    ddr4_rtl_0_act_n => act_n,
    ddr4_rtl_0_adr => adr,
    ddr4_rtl_0_ba => ba,
    ddr4_rtl_0_bg => bg,
    ddr4_rtl_0_ck_c => ck_c,
    ddr4_rtl_0_ck_t => ck_t,
    ddr4_rtl_0_cke => cke,
    ddr4_rtl_0_cs_n => cs_n,
    ddr4_rtl_0_dm_n => dm_n,
    ddr4_rtl_0_dq => dq,
    ddr4_rtl_0_dqs_c => dqs_c,
    ddr4_rtl_0_dqs_t => dqs_t,
    ddr4_rtl_0_odt => odt,
    ddr4_rtl_0_reset_n => reset_n,
    reset_rtl_0 => reset_rtl
  );


    bb2_eb1_DDR4_ACT_N <= act_n;


    bb2_eb1_DDR4_A0 <= adr(0);
    bb2_eb1_DDR4_A1<= adr(1);
    bb2_eb1_DDR4_A2<= adr(2);
    bb2_eb1_DDR4_A3<= adr(3);
    bb2_eb1_DDR4_A4<= adr(4);
    bb2_eb1_DDR4_A5<= adr(5);
    bb2_eb1_DDR4_A6<= adr(6);
```

```
bb2_eb1_DDR4_A7<= adr(7);
bb2_eb1_DDR4_A8<= adr(8);
bb2_eb1_DDR4_A9<= adr(9);
bb2_eb1_DDR4_A10_AP<= adr(10);
bb2_eb1_DDR4_A11<= adr(11);
bb2_eb1_DDR4_A12_BC_N<= adr(12);
bb2_eb1_DDR4_A13<= adr(13);
bb2_eb1_DDR4_A14_WE_N<= adr(14);
bb2_eb1_DDR4_A15_CAS_N<= adr(15);
bb2_eb1_DDR4_A16_RAS_N<= adr(16);


bb2_eb1_DDR4_BA0 <= ba(0);
bb2_eb1_DDR4_BA1 <= ba(1);

bb2_eb1_DDR4_BG0 <= bg(0);

bb2_eb1_DDR4_CK_C <= ck_c(0);
bb2_eb1_DDR4_CK_T <= ck_t(0);
bb2_eb1_DDR4_CKE <= cke(0);

bb2_eb1_DDR4_CS_N <= cs_n(0);


bb2_eb1_DDR4_DQ00_DM_DBI_N <= dm_n(0);
bb2_eb1_DDR4_DQ01_DM_DBI_N <= dm_n(1);
bb2_eb1_DDR4_DQ02_DM_DBI_N <= dm_n(2);
bb2_eb1_DDR4_DQ03_DM_DBI_N <= dm_n(3);
bb2_eb1_DDR4_DQ04_DM_DBI_N <= dm_n(4);
bb2_eb1_DDR4_DQ05_DM_DBI_N <= dm_n(5);
bb2_eb1_DDR4_DQ06_DM_DBI_N <= dm_n(6);
bb2_eb1_DDR4_DQ07_DM_DBI_N <= dm_n(7);


bb2_eb1_DDR4_DQ00_D0_DX00 <= dq(0);
bb2_eb1_DDR4_DQ00_D1_DX01 <= dq(1);
bb2_eb1_DDR4_DQ00_D2_DX02 <= dq(2);
bb2_eb1_DDR4_DQ00_D3_DX03 <= dq(3);
bb2_eb1_DDR4_DQ00_D4_DX04 <= dq(4);
bb2_eb1_DDR4_DQ00_D5_DX05 <= dq(5);
bb2_eb1_DDR4_DQ00_D6_DX06 <= dq(6);
bb2_eb1_DDR4_DQ00_D7_DX07 <= dq(7);
bb2_eb1_DDR4_DQ01_D0_DX08 <= dq(8);
bb2_eb1_DDR4_DQ01_D1_DX09 <= dq(9);
bb2_eb1_DDR4_DQ01_D2_DX10 <= dq(10);
bb2_eb1_DDR4_DQ01_D3_DX11 <= dq(11);
bb2_eb1_DDR4_DQ01_D4_DX12 <= dq(12);
bb2_eb1_DDR4_DQ01_D5_DX13 <= dq(13);
bb2_eb1_DDR4_DQ01_D6_DX14 <= dq(14);
bb2_eb1_DDR4_DQ01_D7_DX15 <= dq(15);
bb2_eb1_DDR4_DQ02_D0_DX16 <= dq(16);
bb2_eb1_DDR4_DQ02_D1_DX17 <= dq(17);
bb2_eb1_DDR4_DQ02_D2_DX18 <= dq(18);
bb2_eb1_DDR4_DQ02_D3_DX19 <= dq(19);
bb2_eb1_DDR4_DQ02_D4_DX20 <= dq(20);
bb2_eb1_DDR4_DQ02_D5_DX21 <= dq(21);
bb2_eb1_DDR4_DQ02_D6_DX22 <= dq(22);
bb2_eb1_DDR4_DQ02_D7_DX23 <= dq(23);
bb2_eb1_DDR4_DQ03_D0_DX24 <= dq(24);
bb2_eb1_DDR4_DQ03_D1_DX25 <= dq(25);
bb2_eb1_DDR4_DQ03_D2_DX26 <= dq(26);
bb2_eb1_DDR4_DQ03_D3_DX27 <= dq(27);
bb2_eb1_DDR4_DQ03_D4_DX28 <= dq(28);
```

```
bb2_eb1_DDR4_DQ03_D5_DX29 <= dq(29);
bb2_eb1_DDR4_DQ03_D6_DX30 <= dq(30);
bb2_eb1_DDR4_DQ03_D7_DX31 <= dq(31);
bb2_eb1_DDR4_DQ04_D0_DX32 <= dq(32);
bb2_eb1_DDR4_DQ04_D1_DX33 <= dq(33);
bb2_eb1_DDR4_DQ04_D2_DX34 <= dq(34);
bb2_eb1_DDR4_DQ04_D3_DX35 <= dq(35);
bb2_eb1_DDR4_DQ04_D4_DX36 <= dq(36);
bb2_eb1_DDR4_DQ04_D5_DX37 <= dq(37);
bb2_eb1_DDR4_DQ04_D6_DX38 <= dq(38);
bb2_eb1_DDR4_DQ04_D7_DX39 <= dq(39);
bb2_eb1_DDR4_DQ05_D0_DX40 <= dq(40);
bb2_eb1_DDR4_DQ05_D1_DX41 <= dq(41);
bb2_eb1_DDR4_DQ05_D2_DX42 <= dq(42);
bb2_eb1_DDR4_DQ05_D3_DX43 <= dq(43);
bb2_eb1_DDR4_DQ05_D4_DX44 <= dq(44);
bb2_eb1_DDR4_DQ05_D5_DX45 <= dq(45);
bb2_eb1_DDR4_DQ05_D6_DX46 <= dq(46);
bb2_eb1_DDR4_DQ05_D7_DX47 <= dq(47);
bb2_eb1_DDR4_DQ06_D0_DX48 <= dq(48);
bb2_eb1_DDR4_DQ06_D1_DX49 <= dq(49);
bb2_eb1_DDR4_DQ06_D2_DX50 <= dq(50);
bb2_eb1_DDR4_DQ06_D3_DX51 <= dq(51);
bb2_eb1_DDR4_DQ06_D4_DX52 <= dq(52);
bb2_eb1_DDR4_DQ06_D5_DX53 <= dq(53);
bb2_eb1_DDR4_DQ06_D6_DX54 <= dq(54);
bb2_eb1_DDR4_DQ06_D7_DX55 <= dq(55);
bb2_eb1_DDR4_DQ07_D0_DX56 <= dq(56);
bb2_eb1_DDR4_DQ07_D1_DX57 <= dq(57);
bb2_eb1_DDR4_DQ07_D2_DX58 <= dq(58);
bb2_eb1_DDR4_DQ07_D3_DX59 <= dq(59);
bb2_eb1_DDR4_DQ07_D4_DX60 <= dq(60);
bb2_eb1_DDR4_DQ07_D5_DX61 <= dq(61);
bb2_eb1_DDR4_DQ07_D6_DX62 <= dq(62);
bb2_eb1_DDR4_DQ07_D7_DX63 <= dq(63);


bb2_eb1_DDR4_DQ00_DQS_C <= dqs_c(0);
bb2_eb1_DDR4_DQ00_DQS_T <= dqs_t(0);
bb2_eb1_DDR4_DQ01_DQS_C <= dqs_c(1);
bb2_eb1_DDR4_DQ01_DQS_T <= dqs_t(1);
bb2_eb1_DDR4_DQ02_DQS_C <= dqs_c(2);
bb2_eb1_DDR4_DQ02_DQS_T <= dqs_t(2);
bb2_eb1_DDR4_DQ03_DQS_C <= dqs_c(3);
bb2_eb1_DDR4_DQ03_DQS_T <= dqs_t(3);
bb2_eb1_DDR4_DQ04_DQS_C <= dqs_c(4);
bb2_eb1_DDR4_DQ04_DQS_T <= dqs_t(4);
bb2_eb1_DDR4_DQ05_DQS_C <= dqs_c(5);
bb2_eb1_DDR4_DQ05_DQS_T <= dqs_t(5);
bb2_eb1_DDR4_DQ06_DQS_C <= dqs_c(6);
bb2_eb1_DDR4_DQ06_DQS_T <= dqs_t(6);
bb2_eb1_DDR4_DQ07_DQS_C <= dqs_c(7);
bb2_eb1_DDR4_DQ07_DQS_T <= dqs_t(7);

bb2_eb1_DDR4_ODT <= odt(0);

bb2_eb1_DDR4_RESET_N <= reset_n;

clk_n <= bb2_eb1_CLK_IN_N;
clk_p <= bb2_eb1_CLK_IN_P;
reset_rtl <= '0' ;
```

```
    --LEDS
    LED_BLUE2 <= '0';
    LED_GREEN2 <= '0';
    LED_RED2 <= '0';
    LED_YELLOW2<= '0';
    LED_BLUE <= '0';
    LED_GREEN <= '0';


    bb2_eb1_LED_BLUE1    <= '0';
    bb2_eb1_LED_BLUE2    <= '0';
    bb2_eb1_LED_GREEN1   <= '0';
    bb2_eb1_LED_GREEN2   <= '0';
    bb2_eb1_LED_RED1     <= '0';
    bb2_eb1_LED_RED2     <= '0';
    bb2_eb1_LED_YELLOW1 <= '0';
    bb2_eb1_LED_YELLOW2 <= '0';


end beh;
```

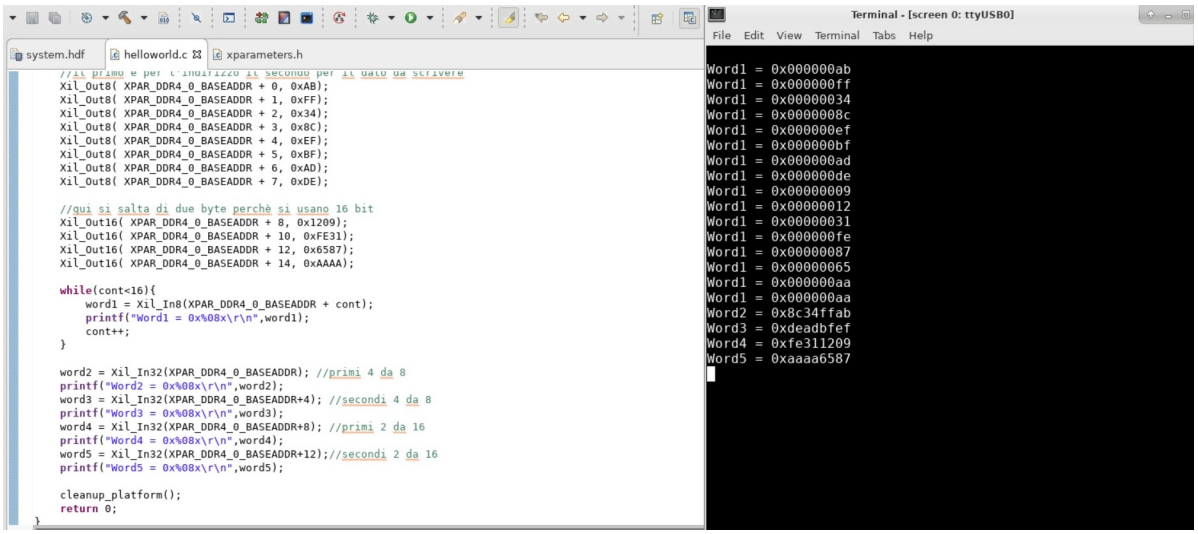

Figure C.11. Block diagram of the test_DDR design.

Listing C.17. helloworld.c C program

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xil_io.h"


int main()
{
    init_platform();

    int word1;  //int is 32 bits
```

```
    int word2;
    int word3;
    int word4;
    int word5;
    int cont=0;

    //writing operation is performed  starting from the base address
  //and adding the required number of bits.
    //for example, if the write is on 8 bits, for next load 1
  //byte have to be added to the address.
    //first parameter is for the address while the second for value
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 0, 0xAB);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 1, 0xFF);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 2, 0x34);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 3, 0x8C);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 4, 0xEF);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 5, 0xBF);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 6, 0xAD);
    Xil_Out8( XPAR_DDR4_0_BASEADDR + 7, 0xDE);

  //here the jump of 2 bytes since is performed a write of 16 bits
    Xil_Out16( XPAR_DDR4_0_BASEADDR + 8, 0x1209);
    Xil_Out16( XPAR_DDR4_0_BASEADDR + 10, 0xFE31);
    Xil_Out16( XPAR_DDR4_0_BASEADDR + 12, 0x6587);
    Xil_Out16( XPAR_DDR4_0_BASEADDR + 14, 0xAAAA);

    while(cont<16){
      word1 = Xil_In8(XPAR_DDR4_0_BASEADDR + cont);
      printf("Word1 = 0x%08x\r\n",word1);
      cont++;
    }

    word2 = Xil_In32(XPAR_DDR4_0_BASEADDR); //firt 4 of 8
    printf("Word2 = 0x%08x\r\n",word2);
    word3 = Xil_In32(XPAR_DDR4_0_BASEADDR+4); //second 4 of 8
    printf("Word3 = 0x%08x\r\n",word3);
    word4 = Xil_In32(XPAR_DDR4_0_BASEADDR+8); //first 2 of 16
    printf("Word4 = 0x%08x\r\n",word4);
    word5 = Xil_In32(XPAR_DDR4_0_BASEADDR+12);//second 2 of 16
    printf("Word5 = 0x%08x\r\n",word5);

    cleanup_platform();
    return 0;
}
```

C.7 Test of Custom slave device

Listing C.18. mult.vhd multiplier file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mult is
    Port ( clk : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (15 downto 0);
           b : in STD_LOGIC_VECTOR (15 downto 0);
         sel : in STD_LOGIC_VECTOR(1 downto 0);
           calc_out : out STD_LOGIC_VECTOR (31 downto 0));
end mult;

architecture Behavioral of mult is


signal first,second : signed(15 downto 0);
signal result : signed(31 downto 0);


begin



    process (clk)
    begin
        if clk'event and clk = '1' then


            case sel is
                when b"10" =>
                        result <= (first * second);

                when others =>
                        result <= (others => '0');
            end case;
        end if;
    end process;

first <= signed(a);
second <= signed(b);
calc_out <= std_logic_vector(result);



end Behavioral;
```

Listing C.19. TOP.vhd TOP file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Library UNISIM;
use UNISIM.vcomponents.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TOP is
port (
   LED_BLUE : out std_logic;
   LED_BLUE2 : out std_logic;
   LED_GREEN : out std_logic;
   LED_GREEN2 : out std_logic;
   LED_RED : out std_logic;
   LED_RED2 : out std_logic;
   LED_YELLOW : out std_logic;
   LED_YELLOW2 : out std_logic

    );
end TOP;

architecture Behavioral of TOP is

component design_xczu19eg_wrapper is
  port (
    gpio_rtl_0_tri_o : out STD_LOGIC_VECTOR ( 7 downto 0 )
  );
end  component;

signal LED_strip : std_logic_vector (7 downto 0);

begin

    processore : design_xczu19eg_wrapper port map (gpio_rtl_0_tri_o => LED_strip);

   LED_BLUE <= LED_strip(3);
   LED_GREEN <= LED_strip(0);
   LED_RED <= LED_strip(1);
   LED_YELLOW <= LED_strip(2);
   LED_BLUE2 <= LED_strip(7);
   LED_GREEN2 <= LED_strip(4);
   LED_RED2 <= LED_strip(5);
   LED_YELLOW2 <= LED_strip(6);


end Behavioral;
```

Listing C.20. mymultiplicator_v1_0.vhd multiplier IP VHDL file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mymultiplicator_v1_0 is
  generic (
    -- Users to add parameters here

    -- User parameters ends
    -- Do not modify the parameters beyond this line


    -- Parameters of Axi Slave Bus Interface S00_AXI
    C_S00_AXI_DATA_WIDTH   : integer := 32;
    C_S00_AXI_ADDR_WIDTH   : integer := 4
  );
  port (
```

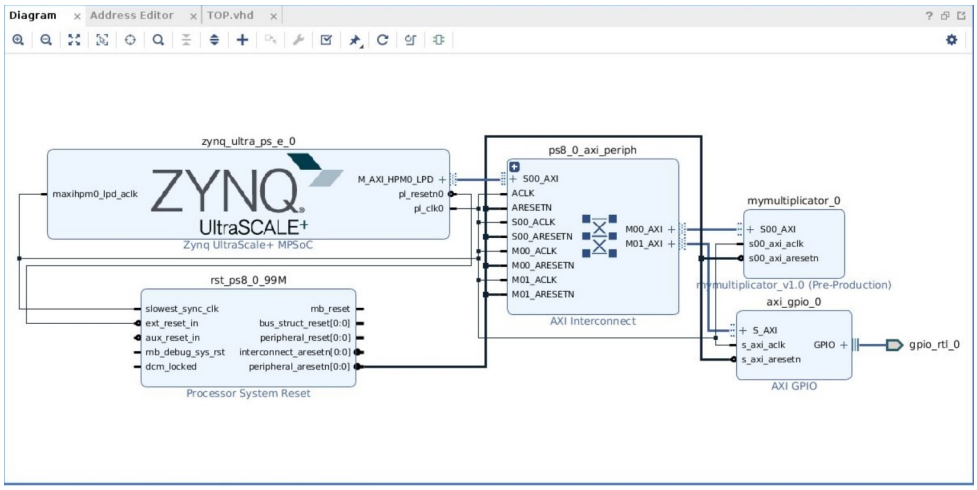

Figure C.12. Block diagram of the test_calc design.

```
    -- Users to add ports here

    -- User ports ends
    -- Do not modify the ports beyond this line


    -- Ports of Axi Slave Bus Interface S00_AXI
    s00_axi_aclk    : in std_logic;
    s00_axi_aresetn : in std_logic;
    s00_axi_awaddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
    s00_axi_awprot  : in std_logic_vector(2 downto 0);
    s00_axi_awvalid : in std_logic;
    s00_axi_awready : out std_logic;
    s00_axi_wdata   : in std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    s00_axi_wstrb   : in std_logic_vector((C_S00_AXI_DATA_WIDTH/8)-1 downto 0);
    s00_axi_wvalid  : in std_logic;
    s00_axi_wready  : out std_logic;
    s00_axi_bresp   : out std_logic_vector(1 downto 0);
    s00_axi_bvalid  : out std_logic;
    s00_axi_bready  : in std_logic;
    s00_axi_araddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
    s00_axi_arprot  : in std_logic_vector(2 downto 0);
    s00_axi_arvalid : in std_logic;
    s00_axi_arready : out std_logic;
    s00_axi_rdata   : out std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    s00_axi_rresp   : out std_logic_vector(1 downto 0);
    s00_axi_rvalid  : out std_logic;
    s00_axi_rready  : in std_logic
  );
end mymultiplicator_v1_0;

architecture arch_imp of mymultiplicator_v1_0 is

  -- component declaration
  component mymultiplicator_v1_0_S00_AXI is
    generic (
    C_S_AXI_DATA_WIDTH  : integer := 32;
    C_S_AXI_ADDR_WIDTH  : integer := 4
    );
```

```
    port (
    S_AXI_ACLK   : in std_logic;
    S_AXI_ARESETN : in std_logic;
    S_AXI_AWADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_AWPROT  : in std_logic_vector(2 downto 0);
    S_AXI_AWVALID : in std_logic;
    S_AXI_AWREADY : out std_logic;
    S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
    S_AXI_WVALID  : in std_logic;
    S_AXI_WREADY  : out std_logic;
    S_AXI_BRESP : out std_logic_vector(1 downto 0);
    S_AXI_BVALID  : out std_logic;
    S_AXI_BREADY  : in std_logic;
    S_AXI_ARADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_ARPROT  : in std_logic_vector(2 downto 0);
    S_AXI_ARVALID : in std_logic;
    S_AXI_ARREADY : out std_logic;
    S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_RRESP : out std_logic_vector(1 downto 0);
    S_AXI_RVALID  : out std_logic;
    S_AXI_RREADY  : in std_logic
    );
  end component mymultiplicator_v1_0_S00_AXI;

begin

-- Instantiation of Axi Bus Interface S00_AXI
mymultiplicator_v1_0_S00_AXI_inst : mymultiplicator_v1_0_S00_AXI
  generic map (
    C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
    C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
  )
  port map (
    S_AXI_ACLK  => s00_axi_aclk,
    S_AXI_ARESETN => s00_axi_aresetn,
    S_AXI_AWADDR  => s00_axi_awaddr,
    S_AXI_AWPROT  => s00_axi_awprot,
    S_AXI_AWVALID => s00_axi_awvalid,
    S_AXI_AWREADY => s00_axi_awready,
    S_AXI_WDATA => s00_axi_wdata,
    S_AXI_WSTRB => s00_axi_wstrb,
    S_AXI_WVALID  => s00_axi_wvalid,
    S_AXI_WREADY  => s00_axi_wready,
    S_AXI_BRESP => s00_axi_bresp,
    S_AXI_BVALID  => s00_axi_bvalid,
    S_AXI_BREADY  => s00_axi_bready,
    S_AXI_ARADDR  => s00_axi_araddr,
    S_AXI_ARPROT  => s00_axi_arprot,
    S_AXI_ARVALID => s00_axi_arvalid,
    S_AXI_ARREADY => s00_axi_arready,
    S_AXI_RDATA => s00_axi_rdata,
    S_AXI_RRESP => s00_axi_rresp,
    S_AXI_RVALID  => s00_axi_rvalid,
    S_AXI_RREADY  => s00_axi_rready
  );

  -- Add user logic here

  -- User logic ends

end arch_imp;
```

Listing C.21. mymultiplicator_v1_0_S00_AXI.vhd multiplier AXI implementation VHDL file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mymultiplicator_v1_0_S00_AXI is
  generic (
    -- Users to add parameters here

    -- User parameters ends
    -- Do not modify the parameters beyond this line

    -- Width of S_AXI data bus
    C_S_AXI_DATA_WIDTH  : integer := 32;
    -- Width of S_AXI address bus
    C_S_AXI_ADDR_WIDTH  : integer := 4
  );
  port (
    -- Users to add ports here

    -- User ports ends
    -- Do not modify the ports beyond this line

    -- Global Clock Signal
    S_AXI_ACLK  : in std_logic;
    -- Global Reset Signal. This Signal is Active LOW
    S_AXI_ARESETN : in std_logic;
    -- Write address (issued by master, acceped by Slave)
    S_AXI_AWADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    -- Write channel Protection type. This signal indicates the
        -- privilege and security level of the transaction, and whether
        -- the transaction is a data access or an instruction access.
    S_AXI_AWPROT  : in std_logic_vector(2 downto 0);
    -- Write address valid. This signal indicates that the master signaling
        -- valid write address and control information.
    S_AXI_AWVALID : in std_logic;
    -- Write address ready. This signal indicates that the slave is ready
        -- to accept an address and associated control signals.
    S_AXI_AWREADY : out std_logic;
    -- Write data (issued by master, acceped by Slave)
    S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    -- Write strobes. This signal indicates which byte lanes hold
        -- valid data. There is one write strobe bit for each eight
        -- bits of the write data bus.
    S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
    -- Write valid. This signal indicates that valid write
        -- data and strobes are available.
    S_AXI_WVALID  : in std_logic;
    -- Write ready. This signal indicates that the slave
        -- can accept the write data.
    S_AXI_WREADY  : out std_logic;
    -- Write response. This signal indicates the status
        -- of the write transaction.
    S_AXI_BRESP : out std_logic_vector(1 downto 0);
    -- Write response valid. This signal indicates that the channel
        -- is signaling a valid write response.
    S_AXI_BVALID  : out std_logic;
    -- Response ready. This signal indicates that the master
        -- can accept a write response.
    S_AXI_BREADY  : in std_logic;
    -- Read address (issued by master, acceped by Slave)
    S_AXI_ARADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
```

```
    -- Protection type. This signal indicates the privilege
        -- and security level of the transaction, and whether the
        -- transaction is a data access or an instruction access.
    S_AXI_ARPROT  : in std_logic_vector(2 downto 0);
    -- Read address valid. This signal indicates that the channel
        -- is signaling valid read address and control information.
    S_AXI_ARVALID : in std_logic;
    -- Read address ready. This signal indicates that the slave is
        -- ready to accept an address and associated control signals.
    S_AXI_ARREADY : out std_logic;
    -- Read data (issued by slave)
    S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    -- Read response. This signal indicates the status of the
        -- read transfer.
    S_AXI_RRESP : out std_logic_vector(1 downto 0);
    -- Read valid. This signal indicates that the channel is
        -- signaling the required read data.
    S_AXI_RVALID  : out std_logic;
    -- Read ready. This signal indicates that the master can
        -- accept the read data and response information.
    S_AXI_RREADY  : in std_logic
  );
end mymultiplicator_v1_0_S00_AXI;

architecture arch_imp of mymultiplicator_v1_0_S00_AXI is

  -- AXI4LITE signals
  signal axi_awaddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_awready  : std_logic;
  signal axi_wready : std_logic;
  signal axi_bresp  : std_logic_vector(1 downto 0);
  signal axi_bvalid : std_logic;
  signal axi_araddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_arready  : std_logic;
  signal axi_rdata  : std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal axi_rresp  : std_logic_vector(1 downto 0);
  signal axi_rvalid : std_logic;

  -- Example-specific design signals
  -- local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
  -- ADDR_LSB is used for addressing 32/64 bit registers/memories
  -- ADDR_LSB = 2 for 32 bits (n downto 2)
  -- ADDR_LSB = 3 for 64 bits (n downto 3)
  constant ADDR_LSB  : integer := (C_S_AXI_DATA_WIDTH/32)+ 1;
  constant OPT_MEM_ADDR_BITS : integer := 1;
  ------------------------------------------------
  ---- Signals for user logic register space example
  ------------------------------------------------
  ---- Number of Slave Registers 4
  signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal slv_reg_rden : std_logic;
  signal slv_reg_wren : std_logic;
  signal reg_data_out :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal byte_index : integer;
  signal aw_en  : std_logic;

  signal calculator_out : std_logic_vector (31 downto 0);

  component mult is
    Port ( clk : in STD_LOGIC;
```

```
            a : in STD_LOGIC_VECTOR (15 downto 0);
            b : in STD_LOGIC_VECTOR (15 downto 0);
          sel : in STD_LOGIC_VECTOR(1 downto 0);
            calc_out : out STD_LOGIC_VECTOR (31 downto 0));
    end component;

begin
  -- I/O Connections assignments

  S_AXI_AWREADY <= axi_awready;
  S_AXI_WREADY  <= axi_wready;
  S_AXI_BRESP <= axi_bresp;
  S_AXI_BVALID  <= axi_bvalid;
  S_AXI_ARREADY <= axi_arready;
  S_AXI_RDATA <= axi_rdata;
  S_AXI_RRESP <= axi_rresp;
  S_AXI_RVALID  <= axi_rvalid;
  -- Implement axi_awready generation
  -- axi_awready is asserted for one S_AXI_ACLK clock cycle when both
  -- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
  -- de-asserted when reset is low.

  process (S_AXI_ACLK)
  begin
    if rising_edge(S_AXI_ACLK) then
      if S_AXI_ARESETN = '0' then
        axi_awready <= '0';
        aw_en <= '1';
      else
        if (axi_awready = '0' and S_AXI_AWVALID = '1' and S_AXI_WVALID = '1' and
            aw_en = '1') then
          -- slave is ready to accept write address when
          -- there is a valid write address and write data
          -- on the write address and data bus. This design
          -- expects no outstanding transactions.
            axi_awready <= '1';
            aw_en <= '0';
         elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then
            aw_en <= '1';
            axi_awready <= '0';
        else
          axi_awready <= '0';
        end if;
      end if;
    end if;
  end process;

  -- Implement axi_awaddr latching
  -- This process is used to latch the address when both
  -- S_AXI_AWVALID and S_AXI_WVALID are valid.

  process (S_AXI_ACLK)
  begin
    if rising_edge(S_AXI_ACLK) then
      if S_AXI_ARESETN = '0' then
        axi_awaddr <= (others => '0');
      else
        if (axi_awready = '0' and S_AXI_AWVALID = '1' and S_AXI_WVALID = '1' and
            aw_en = '1') then
          -- Write Address latching
          axi_awaddr <= S_AXI_AWADDR;
        end if;
      end if;
```

```
  end if;
end process;

-- Implement axi_wready generation
-- axi_wready is asserted for one S_AXI_ACLK clock cycle when both
-- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
-- de-asserted when reset is low.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_wready <= '0';
    else
      if (axi_wready = '0' and S_AXI_WVALID = '1' and S_AXI_AWVALID = '1' and
          aw_en = '1') then
          -- slave is ready to accept write data when
          -- there is a valid write address and write data
          -- on the write address and data bus. This design
          -- expects no outstanding transactions.
          axi_wready <= '1';
      else
        axi_wready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement memory mapped register select and write logic generation
-- The write data is accepted and written to memory mapped registers when
-- axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write
    strobes are used to
-- select byte enables of slave registers while writing.
-- These registers are cleared when reset (active low) is applied.
-- Slave register write enable is asserted when valid address and data are
    available
-- and the slave is ready to accept the write address and write data.
slv_reg_wren <= axi_wready and S_AXI_WVALID and axi_awready and S_AXI_AWVALID ;

process (S_AXI_ACLK)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
      slv_reg2 <= (others => '0');
      slv_reg3 <= (others => '0');
    else
      loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
      if (slv_reg_wren = '1') then
        case loc_addr is
          when b"00" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 0
                slv_reg0(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"01" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
```

```
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 1
                slv_reg1(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"10" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 2
                slv_reg2(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"11" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 3
                slv_reg3(byte_index*8+7 downto byte_index*8) <=
                    S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when others =>
            slv_reg0 <= slv_reg0;
            slv_reg1 <= slv_reg1;
            slv_reg2 <= slv_reg2;
            slv_reg3 <= slv_reg3;
        end case;
      end if;
    end if;
  end if;
end process;

-- Implement write response logic generation
-- The write response and response valid signals are asserted by the slave
-- when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
-- This marks the acceptance of address and indicates the status of
-- write transaction.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_bvalid  <= '0';
      axi_bresp   <= "00"; --need to work more on the responses
    else
      if (axi_awready = '1' and S_AXI_AWVALID = '1' and axi_wready = '1' and
          S_AXI_WVALID = '1' and axi_bvalid = '0'  ) then
        axi_bvalid <= '1';
        axi_bresp  <= "00";
      elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then   --check if bready
          is asserted while bvalid is high)
        axi_bvalid <= '0';                                 -- (there is a
          possibility that bready is always asserted high)
      end if;
    end if;
  end if;
end process;

-- Implement axi_arready generation
```

```
-- axi_aready is asserted for one S_AXI_ACLK clock cycle when
-- S_AXI_ARVALID is asserted. axi_awready is
-- de-asserted when reset (active low) is asserted.
-- The read address is also latched when S_AXI_ARVALID is
-- asserted. axi_araddr is reset to zero on reset assertion.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_aready <= '0';
      axi_araddr  <= (others => '1');
    else
      if (axi_aready = '0' and S_AXI_ARVALID = '1') then
        -- indicates that the slave has acceped the valid read address
        axi_aready <= '1';
        -- Read Address latching
        axi_araddr  <= S_AXI_ARADDR;
      else
        axi_aready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement axi_arvalid generation
-- axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
-- S_AXI_ARVALID and axi_aready are asserted. The slave registers
-- data are available on the axi_rdata bus at this instance. The
-- assertion of axi_rvalid marks the validity of read data on the
-- bus and axi_rresp indicates the status of read transaction.axi_rvalid
-- is deasserted on reset (active low). axi_rresp and axi_rdata are
-- cleared to zero on reset (active low).
process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_rvalid <= '0';
      axi_rresp  <= "00";
    else
      if (axi_aready = '1' and S_AXI_ARVALID = '1' and axi_rvalid = '0') then
        -- Valid read data is available at the read data bus
        axi_rvalid <= '1';
        axi_rresp  <= "00"; -- 'OKAY' response
      elsif (axi_rvalid = '1' and S_AXI_RREADY = '1') then
        -- Read data is accepted by the master
        axi_rvalid <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement memory mapped register select and read logic generation
-- Slave register read enable is asserted when valid address is available
-- and the slave is ready to accept the read address.
slv_reg_rden <= axi_aready and S_AXI_ARVALID and (not axi_rvalid) ;

process (slv_reg0, slv_reg1, slv_reg2, calculator_out, axi_araddr,
    S_AXI_ARESETN, slv_reg_rden)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
```

```
      case loc_addr is
        when b"00" =>
          reg_data_out <= slv_reg0;
        when b"01" =>
          reg_data_out <= slv_reg1;
        when b"10" =>
          reg_data_out <= slv_reg2;
        when b"11" =>
          reg_data_out <= calculator_out;
        when others =>
          reg_data_out  <= (others => '0');
      end case;
  end process;

  -- Output register or memory read data
  process( S_AXI_ACLK ) is
  begin
    if (rising_edge (S_AXI_ACLK)) then
      if ( S_AXI_ARESETN = '0' ) then
        axi_rdata  <= (others => '0');
      else
        if (slv_reg_rden = '1') then
          -- When there is a valid read address (S_AXI_ARVALID) with
          -- acceptance of read address by the slave (axi_aready),
          -- output the read dada
          -- Read address mux
            axi_rdata <= reg_data_out;     -- register read data
        end if;
      end if;
    end if;
  end process;


  -- Add user logic here

    calculator_0 : mult
                 port map (
                  clk => S_AXI_ACLK,
                    a => slv_reg0(15 downto 0),
                    b => slv_reg1(15 downto 0),
                  sel => slv_reg2(1 downto 0),
                    calc_out => calculator_out);
  -- User logic ends

end arch_imp;
```

Listing C.22. multiplier.c C standalone program

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <xgpio.h>
#include "xil_io.h"
#include "xparameters.h"
#include "sleep.h"


int main()
{

  int a,b,sel,result=0;
  int test_a=0,test_b=0,test_sel=0;
```

```
a=35;
b=234;
sel=2;


XGpio led_blinker;


XGpio_Initialize(&led_blinker,XPAR_AXI_GPIO_0_DEVICE_ID); //initialize output
    xgpio variable

XGpio_SetDataDirection(&led_blinker,1,0x0); //set first channel buffer tristate
    to output

init_platform();

//turn off all leds
XGpio_DiscreteWrite(&led_blinker,1,0b00000000);


//evaluate multiplication

//select to do division
Xil_Out32( XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR + 8, sel);

//write first operand
Xil_Out32( XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR + 0, a);
//write second operand
Xil_Out32( XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR + 4, b);

//read operand to be sure of communication
  test_a = Xil_In32(XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR);
  printf("a = %d\r\n",a);
  printf("test_a = %d\r\n",test_a);
  printf("a = 0x%08x\r\n",a);
  printf("test_a = 0x%08x\r\n",test_a);

  printf("\r\n");
  printf("\r\n");

  test_b = Xil_In32(XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR+4);
printf("b = %d\r\n",b);
printf("test_b = %d\r\n",test_b);
printf("b = 0x%08x\r\n",b);
printf("test_b = 0x%08x\r\n",test_b);


printf("\r\n");
printf("\r\n");

test_sel = Xil_In32(XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR+8);
printf("sel = %d\r\n",sel);
printf("test_sel = %d\r\n",test_sel);
printf("sel = 0x%08x\r\n",sel);
printf("test_sel = 0x%08x\r\n",test_sel);


//read result of multiplication
result = Xil_In32(XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR+12);
printf("result = %d\r\n",result);
```

```
  //control result on leds

  if (result<10){

    XGpio_DiscreteWrite(&led_blinker,1,0b00000000);
    }
  if (result>=10 && result<100){

    XGpio_DiscreteWrite(&led_blinker,1,0b00000001);
    }

  if (result>=100 && result<1000){

    XGpio_DiscreteWrite(&led_blinker,1,0b00000010);
    }
  if (result>=1000 && result<10000){

    XGpio_DiscreteWrite(&led_blinker,1,0b00000100);
    }
  if (result>=10000 && result<100000){

    XGpio_DiscreteWrite(&led_blinker,1,0b00001000);
    }
  if (result>=100000 && result<1000000){

      XGpio_DiscreteWrite(&led_blinker,1,0b00010000);
      }
  if (result>=1000000 && result<10000000){

      XGpio_DiscreteWrite(&led_blinker,1,0b00100000);
      }
  if (result>=10000000 && result<100000000){

      XGpio_DiscreteWrite(&led_blinker,1,0b01000000);
      }
  if (result>=100000000 && result<1000000000){

      XGpio_DiscreteWrite(&led_blinker,1,0b10000000);
      }


  usleep(2000000);
    cleanup_platform();
    return 0;
}
```

Listing C.23. codemult.c C OS program

```
/*****************************************************************************/
/*                          INCLUDE FILES                                    */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>


/*****************************************************************************/
/*                            DEFINES                                        */
/*****************************************************************************/
```

```
#define GPIO_BASE_ADDR  0x41200000
#define XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR 0x80000000
#define XPAR_GPIO_0_BASEADDR 0x80001000


/************************************************************************/
/*                                MAIN                                  */
/************************************************************************/


int main(void)
{
  int      a=0,b=0,sel=2,result=0,cont=0;
  int      test_a=0,test_b=0,test_sel=0;
  int      fd;
  void     *ptr_gpio,*ptr_mult;
  unsigned  page_addr_gpio,page_addr_mult;
  unsigned  page_offset_gpio,page_offset_mult;
  unsigned  page_size = sysconf(_SC_PAGESIZE);


  fd = open("/dev/mem",O_RDWR);

  page_addr_gpio    = (XPAR_GPIO_0_BASEADDR & ~(page_size-1));
  page_offset_gpio = XPAR_GPIO_0_BASEADDR - page_addr_gpio;

  page_addr_mult    = (XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR & ~(page_size-1));
  page_offset_mult = XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR - page_addr_mult;

  ptr_gpio =
      mmap(NULL,page_size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,(XPAR_GPIO_0_BASEADDR
      & ~(page_size-1)));
  ptr_mult = mmap(NULL,page_size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,
              (XPAR_MYMULTIPLICATOR_0_S00_AXI_BASEADDR & ~(page_size-1)));



  //clear all LED
  //write to memory address of gpio to turn all led off
  *((unsigned *)(ptr_gpio))= 0;


  while(cont<5){

    printf("Select first operand: ");
    scanf("%d", &a);
    printf("\r\nSelect second operand: ");
    scanf("%d", &b);

    //select to do division
    *((unsigned *)(ptr_mult+8))= sel;

    //write first operand
    *((unsigned *)(ptr_mult))= a;

    //write second operand
    *((unsigned *)(ptr_mult+4))= b;


    //read operand to be sure of communication
    test_sel = *((unsigned *)(ptr_mult+8));
```

```
    test_a = *((unsigned *)(ptr_mult));
    test_b = *((unsigned *)(ptr_mult+4));

    printf("a = %d\r\n",a);
         printf("test_a = %d\r\n",test_a);
         printf("a = 0x%08x\r\n",a);
         printf("test_a = 0x%08x\r\n",test_a);

    printf("\r\n");
         printf("\r\n");

    printf("b = %d\r\n",b);
    printf("test_b = %d\r\n",test_b);
    printf("b = 0x%08x\r\n",b);
    printf("test_b = 0x%08x\r\n",test_b);

    printf("\r\n");
         printf("\r\n");

    printf("sel = %d\r\n",sel);
    printf("test_sel = %d\r\n",test_sel);
    printf("sel = 0x%08x\r\n",sel);
    printf("test_sel = 0x%08x\r\n",test_sel);

    printf("\r\n");
         printf("\r\n");

    //read result of multiplication
    result = *((unsigned *)(ptr_mult+12));

    printf("result = %d\r\n",result);

    printf("\r\n");
         printf("\r\n");

    cont++;

  }

  //remove memory allocation
  munmap(ptr_gpio,page_size);
  munmap(ptr_mult,page_size);
  //close file
  close(fd);


  return 0;

}
```

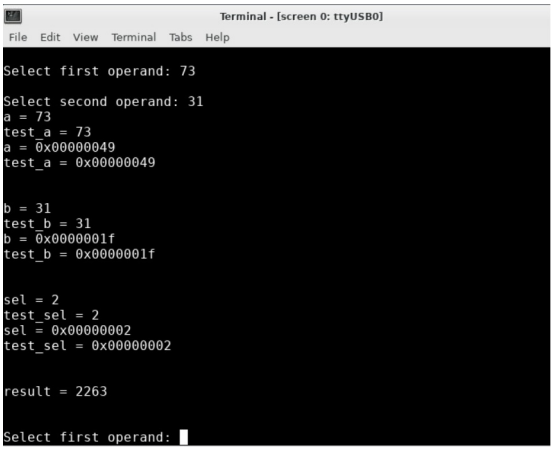

Figure C.13. Test on OS of the multiplier with positive numbers.

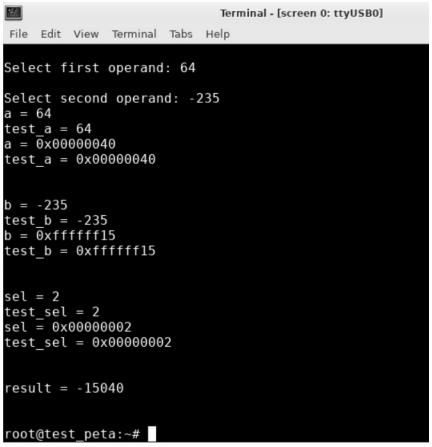

Figure C.14. Test on OS of the multiplier with negative numbers.

Appendix D

Configuration file generated by ProFPGA

Listing D.1. main.c C code file with which the dataset is generated

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "KeccakPRGWidth1600.h"

#define KECCAK_CAPACITY 576
#define BIT_BUFFER_LENGTH 32
#define KEY_LENGTH 32



unsigned char public_key[KEY_LENGTH] = "publickeypublickeypublickeypubli";
unsigned char iv_vector[KEY_LENGTH] = "initializationvectorinitializati";
unsigned char deactivate_encryption = '0';
unsigned char all_key[2*KEY_LENGTH] =
    "publickeypublickeypublickeypubliinitializationvectorinitializati";



//------------------------------------------------------------------
/* run this program using the console pauser or add your own getch,
    system("pause") or input loop */


int conv_ascii(char valore) {

  int veroval=0;

  if(isdigit(valore)){

    veroval = (int)valore - 48;
  }else{

    if(isupper(valore)){

    veroval = (int)valore - 55;
    }else{
```

```
    veroval = (int)valore - 87;
    }

  }


  return veroval;
}


typedef struct byteBuffer{
  int counter;
  int buffer_length;
  unsigned char *bytes;
  KeccakWidth1600_SpongePRG_Instance instance;
} byteBuffer;




int main(int argc, char *argv[]) {

  KeccakWidth1600_SpongePRG_Instance instance;

  char fileByte, encrByte;
  int temp=0,punt=0,i,j,varinput,position,total=0;
  //generate file names
  char inputName[128];
  strcpy(inputName,"new_test_vector/keccak_in");
  char inputExt[128];
  strcpy(inputExt,".txt");
  char inputFilename[128];
  char encryptedFilename[128];
  char origFilename[128];
  char keyFilename[128];
  char keyvhdl[128];
  char vhdlout[128];
  char vhdlinsnake[128];
  char testkey[128];
  strcpy(inputFilename,inputName);
  strcat(inputFilename,inputExt);
  strcpy(encryptedFilename,inputName);
  strcat(encryptedFilename,".encr");
  strcat(encryptedFilename,inputExt);
  strcpy(origFilename,inputName);
  strcat(origFilename,".ORIG");
  strcat(origFilename,inputExt);
  strcpy(keyFilename,inputName);
  strcat(keyFilename,".KEY");
  strcat(keyFilename,inputExt);
  strcpy(keyvhdl,inputName);
  strcat(keyvhdl,".VHDL");
  strcat(keyvhdl,inputExt);
  strcpy(vhdlout,inputName);
  strcat(vhdlout,".VHDLOUT");
  strcat(vhdlout,inputExt);
  strcpy(vhdlinsnake,inputName);
  strcat(vhdlinsnake,".VHDLinverse");
  strcat(vhdlinsnake,inputExt);
  strcpy(testkey,inputName);
  strcat(testkey,".TESTKEY");
```

```
strcat(testkey,inputExt);


//open files
FILE *fp = fopen(inputFilename,"rb");
FILE *encrypted_file = fopen(encryptedFilename,"wab+");
FILE *orig_file = fopen(origFilename,"wab+");
FILE *key_file = fopen(keyFilename,"wab+");
FILE *key_vhdl = fopen(keyvhdl,"wab+");
FILE *key_vhdlout = fopen(vhdlout,"wab+");
FILE *key = fopen(vhdlinsnake,"wab+");

struct byteBuffer buffer;


//per vedere cosa entra

for(i=0 ; i<32 ; i++){

fprintf(key_file,"%d␣␣%c␣␣%x␣--␣",public_key[i],public_key[i],public_key[i]);

}

fprintf(key_file,"\n");
fprintf(key_file,"-----------------------------------------------\n");

for(j=0 ; j<32 ; j++){

fprintf(key_file,"%d␣␣%c␣␣%x␣--␣",iv_vector[j],iv_vector[j],iv_vector[j]);

}



//per il vhdl

//shared_key

for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdl,"%x␣--␣",public_key[varinput]);
}
fprintf(key_vhdl,"\n");

for(varinput=8; varinput<16; varinput++){
  fprintf(key_vhdl,"%x␣--␣",public_key[varinput]);
}
fprintf(key_vhdl,"\n");

for(varinput=16; varinput<24; varinput++){
  fprintf(key_vhdl,"%x␣--␣",public_key[varinput]);
}
fprintf(key_vhdl,"\n");

for(varinput=24; varinput<32; varinput++){
  fprintf(key_vhdl,"%x␣--␣",public_key[varinput]);
}
fprintf(key_vhdl,"\n");

//iv_vector

for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdl,"%x␣--␣",iv_vector[varinput]);
```

```
}
fprintf(key_vhdl,"\n");

for(varinput=8; varinput<16; varinput++){
  fprintf(key_vhdl,"%x -- ",iv_vector[varinput]);
}
fprintf(key_vhdl,"\n");

for(varinput=16; varinput<24; varinput++){
  fprintf(key_vhdl,"%x -- ",iv_vector[varinput]);
}
fprintf(key_vhdl,"\n");

for(varinput=24; varinput<32; varinput++){
  fprintf(key_vhdl,"%x -- ",iv_vector[varinput]);
}
fprintf(key_vhdl,"\n");

//per il vhdl

//shared_key_snake

for(varinput=7; varinput>=0; varinput--){
  fprintf(key,"%x -- ",public_key[varinput]);
}
fprintf(key,"\n");

for(varinput=15; varinput>=8; varinput--){
  fprintf(key,"%x -- ",public_key[varinput]);
}
fprintf(key,"\n");

for(varinput=23; varinput>=16; varinput--){
  fprintf(key,"%x -- ",public_key[varinput]);
}
fprintf(key,"\n");

for(varinput=31; varinput>=24; varinput--){
  fprintf(key,"%x -- ",public_key[varinput]);
}
fprintf(key,"\n");


//iv_vector

for(varinput=7; varinput>=0; varinput--){
  fprintf(key,"%x -- ",iv_vector[varinput]);
}
fprintf(key,"\n");

for(varinput=15; varinput>=8; varinput--){
  fprintf(key,"%x -- ",iv_vector[varinput]);
}
fprintf(key,"\n");

for(varinput=23; varinput>=16; varinput--){
  fprintf(key,"%x -- ",iv_vector[varinput]);
}
fprintf(key,"\n");

for(varinput=31; varinput>=24; varinput--){
  fprintf(key,"%x -- ",iv_vector[varinput]);
}
```

```
fprintf(key,"\n");



putc(deactivate_encryption,encrypted_file);
putc(deactivate_encryption,orig_file);




buffer.counter = 0;
buffer.buffer_length = BIT_BUFFER_LENGTH;
buffer.bytes = malloc(BIT_BUFFER_LENGTH*sizeof(unsigned char));

//dichiarazione keccak e creazione buffer
//inizializzazione buffer
KeccakWidth1600_SpongePRG_Initialize(&buffer.instance, KECCAK_CAPACITY);
//feed con chiave nota
KeccakWidth1600_SpongePRG_Feed(&buffer.instance, public_key, KEY_LENGTH);
//write the session key in the encrypted file
fwrite(iv_vector, 1, KEY_LENGTH, encrypted_file);
putc('\n',encrypted_file);
fwrite(iv_vector, 1, KEY_LENGTH, orig_file);
putc('\n',orig_file);
//feed con chiave arbitraria
KeccakWidth1600_SpongePRG_Feed(&buffer.instance, iv_vector, KEY_LENGTH);
//KeccakWidth1600_SpongePRG_Feed(&buffer.instance, all_key, 2*KEY_LENGTH);
//fetch di richiesta numeri
KeccakWidth1600_SpongePRG_Fetch(&buffer.instance, buffer.bytes,
    buffer.buffer_length);


    //vedere cosa c'e' nel buffer di output

    fprintf(key_vhdlout,"prima chiamata di fetch, restituisce 32 char\n");

    for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdlout,"%x --- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=8; varinput<16; varinput++){
    fprintf(key_vhdlout,"%x --- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=16; varinput<24; varinput++){
    fprintf(key_vhdlout,"%x --- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=24; varinput<32; varinput++){
    fprintf(key_vhdlout,"%x --- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");
```

```
//seconda richiesta numeri
//fetch di richiesta numeri
KeccakWidth1600_SpongePRG_Fetch(&buffer.instance, buffer.bytes,
    buffer.buffer_length);

fprintf(key_vhdlout,"seconda chiamata di fetch, restituisce altri 32
    (64) char\n");

    for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=8; varinput<16; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=16; varinput<24; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=24; varinput<32; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");


//terza richiesta numeri
//fetch di richiesta numeri
KeccakWidth1600_SpongePRG_Fetch(&buffer.instance, buffer.bytes,
    buffer.buffer_length);

fprintf(key_vhdlout,"seconda chiamata di fetch, restituisce altri 32
    (96) char\n");

    for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=8; varinput<16; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=16; varinput<24; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=24; varinput<32; varinput++){
    fprintf(key_vhdlout,"%x -- ",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");


  //quarta richiesta numeri
//fetch di richiesta numeri
KeccakWidth1600_SpongePRG_Fetch(&buffer.instance, buffer.bytes,
    buffer.buffer_length);
```

```
fprintf(key_vhdlout,"seconda␣chiamata␣di␣fetch,␣restituisce␣altri␣32␣
    (128)char\n");

    for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=8; varinput<16; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=16; varinput<24; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=24; varinput<32; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");


//cinque richiesta numeri
//fetch di richiesta numeri
KeccakWidth1600_SpongePRG_Fetch(&buffer.instance, buffer.bytes,
    buffer.buffer_length);


fprintf(key_vhdlout,"seconda␣chiamata␣di␣fetch,␣restituisce␣altri␣32␣
    (160)char\n");
  fprintf(key_vhdlout,"l'ultimo␣di␣prima␣e␣tutti␣questi␣dopo␣sono␣del␣nuovo␣
      vettore\n");
    for(varinput=0; varinput<8; varinput++){
  fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=8; varinput<16; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=16; varinput<24; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");

  for(varinput=24; varinput<32; varinput++){
    fprintf(key_vhdlout,"%x␣--␣",buffer.bytes[varinput]);
  }
  fprintf(key_vhdlout,"\n");



//cancello buffer
KeccakWidth1600_SpongePRG_Forget(&(buffer.instance));
free(buffer.bytes);

fclose(fp);
fclose(orig_file);
fclose(encrypted_file);
```

```
  fclose(key_file);
  fclose(key_vhdl);
  fclose(key_vhdlout);
  fclose(key);

  //fclose(orig_file);
  return 0;
}
```

Bibliography

[1] ANSA, "L'era spaziale ha 60 anni, il 4 ottobre 1957 volava lo Sputnik", ANSA - Sputnik article, October 2019.

[2] FOCUS, Vito Tartamella, *Tutto quello che ci dicono i satelliti*, n.324 October 2019.

[3] EO-ALERT project, `http://www.eo-alert-h2020.eu/`

[4] M. Kerr, S. Cornara, A. Latorre, S. Tonetti, A. Fiengo, T. Guardabrazo, J. I. Bravo, D. Velotto, M. Eineder, S. Jacobsen, H. Breit, O. Koudelka, F. Teschl, E. Magli, T. Bianchi, R. Freddi, M. Benetti, R. Fabrizi, S. Fraile, C. Marcos, *EO-ALERT: NEXT GENERATION SATELLITE PROCESSING CHAIN FOR RAPID CIVIL ALERTS*, 2018.

[5] OHB-I Team. *EO-ALERT-D3.8-Avionics HW-SW ICD*, 2019.

[6] A. Kiely et al., "The new CCSDS standard for low-complexity lossless and near-lossless multispectral and hyperspectral image compression", Proc. of Onboard Payload Data Compression workshop (OBPDC), 2018.

[7] Ian Blanes,Enrico Magli,Joan Serra-Sagristà,"A Tutorial on Image Compression for Optical Space Imaging Systems", September 2014.

[8] CCSDS, "Low-complexity lossless and near-lossless multispectral and hyperspectral image compression",CCSDS 123.0-B-2 blue book, February 2019.

[9] Tiziano Bianchi, Tomas Bjorklund, Enrico Magli, Maurizio Martina, Nicola Prette, Diego Valsesia, "Preliminary on-board compression and data handling design and analysis report", January 2019.

[10] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "The sponge and duplex constructions". September 2019. `https://keccak.team/sponge_duplex.html`.

[11] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "Cryptographic sponge functions",2011.

[12] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "Keccak implementation overview",version 3.1,2011.

[13] Wikipedia, "SHA-3", visited : September 2019 `https://en.`

wikipedia.org/wiki/SHA-3

[14] Wikipedia, "National Institute of Standards and Technology", visited : September 2019 https://it.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology

[15] NIST, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions",NIST-FIPS.202, August 2015, http://dx.doi.org/10.6028/NIST.FIPS.202

[16] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "Note on Keccak parameters and usage", 2015, keccak.noekeon.org/NoteOnKeccakParametersAndUsage.pdf

[17] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications", 2015.

[18] G. V. Assche, "eXtended Keccak Code Package (XKCP)", https://github.com/XKCP/XKCP, visited September 2019.

[19] G. V. Assche, "XKCP documentation", https://github.com/XKCP/XKCP/tree/master/doc, visited September 2019.

[20] OHB-I Team, *EO-ALERT-D3.3-Data chain Functional and Physical Architecture 2*, 2019.

[21] proFPGA, *UD001-3.19-Hardware-UserManual*, 2018.

[22] Xilinx, *zynq-ultrascale-plus-product-selection-guide*, 2018.

[23] proFPGA, *UD003-1.39-ExtensionBoard-DesignGuide*, 2018.

[24] "Problem with mask polling", zedboard-forum, Visited : May 2019.

[25] ProFPGA, "AN021-1.2-ZynqPetaLinux", 2015.

[26] Xilinx,"ug1157-PetaLinux Command Line Reference",2014.

[27] Xilinx,"ug1144-PetaLinux Tools Documentation Reference Guide",2018.

[28] Xilinx,"ug981-PetaLinux Application Development Guide",2014.

[29] Xilinx forum, "XSDK Linux Application and petalinux tools", petalinux-app, Visited: June 2019.

[30] Xilinx, "ug1137-Zynq UltraScale+ MPSoC Software Developer Guide", June 2019.

[31] Xilinx, "Zynq-7000 Example Design - Interrupt handling of PL generated interrupt", https://www.xilinx.com/support/answers/50572.html, Visited: May 2019.

[32] ETHERNETFMC, "RGMII Interface Timing Considerations", https://ethernetfmc.com/rgmii-interface-timing-considerations/ , Visited: June 2019.

[33] Zedboard forum, "Linux /dev/mem accessing switch values", http://zedboard.org/content/linux-devmem-accessing-switch-values, Visited: July 2019.

[34] Sven Andersson, "Zynq design from scratch. Part 41", `http://svenand.blogdrives.com/archive/202.html#.Xd64jNXSJPZ`, Visited: July 2019.

[35] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, "Keccak Team hardware", `https://keccak.team/hardware.html`, Visited: September 2019.

[36] Mohammad A. AlAhmad, Imad Fakhri Alshaikhli, "Broad View of Cryptographic Hash Functions", 2013.

[37] Khaled E. Ahmed, Mohammed M. Farag, "Hardware/Software Co-Design of A Dynamically Configurable SHA-3 System-on-Chip (SoC)", 2015.

[38] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche, "Sponge-based pseudo-random number generators", 2015.

[39] A. Gholipour and S. Mirzakuchaki, "A Pseudorandom Number Generator with KECCAK Hash Function", 2011.