UNIVERSITÀ DEGLI STUDI DI PERUGIA

DEPARTMENT OF ENGINEERING
MASTER DEGREE IN
ELECTRONIC AND TELECOMMUNICATION ENGINEERING

DESIGN AND FPGA IMPLEMENTATION
OF THE TEST EQUIPMENT FOR A DIGITAL
COMMUNICATION SYSTEM OF THE NA62
HIGH-ENERGY PHYSICS EXPERIMENTAL
PLATFORM AT THE CERN SPS

in collaboration with:
Istituto Nazionale di Fisica Nucleare Perugia, INFN PG, IT
Comité Européene pour la Recherche Nucléaire, CERN, CH

STUDENT
MATTIA BARBANERA

SUPERVISOR
PROF. GIANLUCA REALI

CO-SUPERVISOR
DOTT. BRUNO CHECCUCCI

ACADEMIC YEAR 2013-2014
J. S. «What does "commissioning" mean?»

A. C. «Make everything work.»

J. S. «Oh, but that will never happen!»

Nobel prize Jack Steinberger - NA62 spokesman Augusto Ceccucci

20 Nov. 2014 - NA62 meeting
Acknowledgments

I would like to express my gratitude to my supervisor Gianluca Reali for his frankness, for his help during the work and for having introduced me to the INFN.

Moreover, this work would not have been possible without Bruno Checcucci, who introduced me to the world of public and work relationships and helped me during the whole work. I would like to thank all the NA62 group of the INFN Perugia for its support and friendship, especially Mauro Piccini, Cristiano Santoni, Patrizia Cenci and Monica Pepe, and all the people of the NA62 experiment, who cannot all be listed, but particularly Augusto Ceccucci, for his lecture about fate and determination, Dario Soldi, who stood me snoring in Geneva, and Francesco Gonnella, for having introduced me to Quartus III.

Thanks to all my university colleagues Federico, Michele, Gianluca, Martina, and all the others, for these years enjoyed together, but especially to Matteo for driving me crazy since these last two years.

Last, but not least, I owe my family a huge debt of gratitude for allowing me to reach this goal and for its tolerance.
Contents

Acknowledgments.................................................................................................................. 4

Contents .................................................................................................................................. 5

Introduction ............................................................................................................................. 8

Chapter 1. NA62 Experiment ................................................................................................. 10
  1.1 The physics behind the experiment: the Standard Model ............................................. 11
  1.2 NA62 Detectors and Features ..................................................................................... 13
  1.3 Trigger and Data Acquisition System .......................................................................... 15

Chapter 2. Hardware Environment ....................................................................................... 18
  2.1 DS92LV16 Serializer/Deserializer Chip ...................................................................... 18
  2.2 DS15BA101/DS15EA101 Cable Extender Chipset ....................................................... 22
  2.3 Field-Programmable Gate Array .................................................................................. 23
  2.4 DE-2 115 Development Board .................................................................................. 25
  2.4.1 Altera Cyclone IV E FPGA ................................................................................ 27
  2.5 TEL62 boards ........................................................................................................... 30
  2.6 TELDES Project ......................................................................................................... 32
  2.7 InterTEL Project ........................................................................................................ 35

Chapter 3. Software Environment ......................................................................................... 37
  3.1 FPGA Design Flow and Quartus II environment ........................................................ 37
  3.2 VHDL principles ......................................................................................................... 40

Chapter 4. Test Set-Up ........................................................................................................ 44
4.1 TELDES and InterTEL Test Set-ups................................................................. 44
4.2 Tests List and Description.............................................................................. 48

Chapter 5. FPGA Test Equipment......................................................................... 52
5.1 Top Module ................................................................................................. 52
5.2 ROM_circ .................................................................................................... 59
5.3 comparator_top ........................................................................................... 60
  5.3.1 rdata_buf .............................................................................................. 61
  5.3.2 comparator ............................................................................................ 62
  5.3.3 FSM_control .......................................................................................... 65
5.4 Compilation Results ...................................................................................... 69
5.5 In-System Testing ......................................................................................... 70
5.6 InterTEL Integration ..................................................................................... 70

Conclusion ........................................................................................................ 73

List of Figures ..................................................................................................... 75
Bibliography ....................................................................................................... 78

Appendix A ......................................................................................................... 81
Introduction

This thesis work is the result of the collaboration with the Italian "Istituto Nazionale di Fisica Nucleare" (INFN), Perugia section, that started with an internship and lasted about ten months. Such work was in particular developed with the NA62 working group, which is part of the NA62 high-energy physics experimental platform at the European Centre for Nuclear Research, CERN, Geneva (Switzerland). This experiment is the result of an international collaboration with the aim of thoroughly testing the validity of the Standard Model of Particle Physics, verifying the branching ratio of the ultra rare kaon decay $K^+ \rightarrow \pi^+ \nu \bar{\nu}$. The experiment is composed of several working groups coming from many countries, each one developing a dedicated system for the experiment, and forming a multi-cultural and inter-disciplinary environment.

The main aim of the work is the design on FPGA of a test equipment for some electronic boards, in accordance with a list of tests defined in collaboration with Matteo Lupi, a thesis student of the Perugia Department of Engineering who worked at the same projects.

The test equipment is first developed together with Roma Tor Vergata INFN section for the TELDES boards, that are daughter cards for a main board named TEL62. They are both used in the main communication link of the liquid Krypton electromagnetic calorimeter of the experiment, which measures the energy of the particles that pass through it and, in the NA62 case, represents a photon veto. A TELDES board has sixteen channels and each of them equalizes a LVDS data stream and deserializes it into 16-bit 40 MHz words, that are read from a dedicated circuitry in order to recognize the specific events that the experiments is trying to detect. In addition to the design of the above-mentioned equipment, an important part of the work consists in the integration of the circuitries in the harsh environment of the experimental area in Geneva, in collaboration with the INFN Roma Tor Vergata and CERN personnel. The most important issues that showed up during this integration step were the electromagnetic noise and interference, which significantly affected the link.
Besides TELDESs, the test equipment was developed also for InterTEL boards, which are auxiliary cards for the TEL62s. They are used by RICH, LAV and CEDAR detectors, which are supervised, respectively, by INFN Firenze and Perugia, INFN Frascati and University of Birmingham. InterTELs allow the communication between the stations of a detector, improving its performance in particles detection. Also for InterTELs an integration step follows the tests, even though it is not finished yet. Each TEL62 includes a FPGA that has to be exploited for the communication with the InterTELs. In the first step of the integration, carried out together with the detectors working groups, such FPGA was used for the electrical and logical assessment of the InterTELs, configured as stand-alone systems. The second step, which is still ongoing, is the definition of the link topology for each detector and the simulation of the actual work conditions of the InterTELs, i.e. at least with two TEL62 each with an InterTEL plugged on it that communicates with the other. The integration steps of both TELDESs and InterTELs were carried out in Meyrin and Prevessin sites of the CERN.

The work allowed the production of two technical reports, one for the TELDESs and one for the InterTELs, in order to show the results of the tests, that will be published into the Engineering and Equipment Data Management Service (EDMS) of CERN. The collaboration with the LKr working group submitted an article that describes the final design of the calorimeter Level0 trigger system, and was accepted for the proceeding of the 2014 Topical Workshop on Electronics for Particle Physics (TWEPP). Finally, an article that describes the procedure of TELDES tests has been produced and has to be submitted to the IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS and MIC).

In the first chapter, a description of the physics and of the experiment is given, highlighting only the topics of interest. Chapter 2 illustrates the hardware environment, describing the boards used and the devices included in them. For the FPGA programming the Altera Quartus II environment is used, and it is described in Chapter 3, together with an introduction to the VHDL language. Chapter 4 includes the description of the tests that have to be done, while Chapter 5 illustrates the modules that constitute the test equipment, illustrating also the integration step of the InterTEL.
Chapter 1. NA62 Experiment

NA62 is a particle physics experiment installed in the North-Area Facility of the Super Proton Synchrotron (SPS) [1] at CERN (European Centre for Nuclear Research). The SPS is the second largest collider of the CERN accelerators complex, shown in Fig. 1.1, measuring nearly 7 kilometres in circumference and operating at up to 450 GeV; it takes particles from the Proton Synchrotron and accelerates them to provide beams to Large Hadron Collider (LHC) [2], the largest accelerator reaching energies up to 14 TeV, together with the NA61/SHINE, COMPASS and NA62 experiments.

The NA62 experiment [3] [4] aims to measure the rate of the very rare kaon decay $K^+\rightarrow\pi^+\nu\bar{\nu}$ with a precision of 10% by collecting about 100 candidates, in order to perform a decisive test of the Standard Model. Other experiments with the same goals have been conducted, e.g. Brookhaven AGS Experiment 949 [5], but the results were inconclusive due to an uncertainty of about 100%. In order to reach the requested precision, background suppression is mandatory and is achieved by identification and suppression of unwanted particles.
1.1 The Physics behind the Experiment: the Standard Model

The Standard Model of particle physics [6] [7], from now on SM, is a theoretical model describing all the known fundamental particles and the electromagnetic, weak and strong nuclear interactions. Unification is an ongoing theme in physics, with the target of a theory of everything that contains all the particles and all the four fundamental interactions, as in Fig. 1.2. Although SM fails as a complete theory of everything because it does not include the full theory of gravitation or the dark matter, most of its predictions have been experimentally verified with high precision, and it is sometimes regarded as a "theory of almost everything".

![Diagram showing unification process of fundamental interactions](image)

Fig. 1.2 - Unification process of fundamental interactions [8].

As shown in Fig. 1.3, there are two main families of fundamental particles: fermions, the building blocks of matter, and bosons, the carriers of the forces. The two family division is made by looking at spin, charge and mass criteria. All fermions have spin ½, while bosons have integer spin, either 1 or 0.

Each fermion has a corresponding antiparticle, and there are twelve flavours, or kinds, of them, divided into two other categories of six particles, quarks and leptons. All the twelve flavoured particles interact via the electro-weak force, mediated by the W and Z bosons and photons. Fermions are grouped in families of two particles, three for leptons and three for quarks. The defining characteristic of quarks, then, is that they carry colour charge (strong charge), that can be red, green or blue; in fact they also interact via the strong force, by means of gluons.
Fig. 1.3 - Elementary particles and their properties (mass, charge, spin); quarks and leptons are called Fermions and are divided in three generations, formed by the particles of each column of the figure [9].

Because of this, quarks tend to bind together, forming colour-neutral composite particles called hadrons: they can be composed of a quark and an antiquark, forming a meson, or of three quarks, forming a baryon. Examples of these particles are the common protons and neutrons, which are the two baryons with the smallest mass. Quarks also carry a fractional part of the quantized electromagnetic charge, and therefore they interact by means of the electromagnetic force, which is mediated by photons. The composite particles formed by bounded quarks must have integer multiple of elementary charge. Leptons do not carry colour charge, so they do not interact via the strong force; the three neutrinos, then, do not carry electromagnetic charge either, so they interact only via the weak force. Fermions belong to one of three generations, represented by the columns of the Fig. 1.3: combinations of particles belonging to different generations have different life-time, which is decreasing with generations increase.

Bosons are defined as force carriers that mediate strong, weak and electromagnetic interactions. The common characteristic is that they have integer spin, one or zero either. The eight gluons have no mass and no charge and mediate the strong interaction between coloured particles; they are also coloured: six have two colours, one has four and the other has six, so gluons can also interact with themselves. Photons, which are massless and chargless, mediate the electromagnetic force, while the carriers of weak interaction are W and Z bosons: both are massive, but only W has electromagnetic charge. Higgs bosons, finally, are very massive bosons with zero charge and spin, and explain why other particles
are massive, except photons and gluons; because Higgs bosons are massive themselves, they must interact with themselves. The first experimental observation of Higgs bosons dates back to 2012, when the CERN experiments ATLAS and CMS independently reported the detection of this new particle.

The noticeable particles in NA62 are charged K mesons ($K^+$), charged Pi mesons ($\pi^+$), neutrinos and antineutrinos. $K^+$ is a meson composed of up and antistrange (u$\bar{s}$) quarks, with a mass of 493.7 MeV/$c^2$ and a mean life time of $1.24 \cdot 10^{-8}$ seconds; $\pi^+$ is the lighter meson, with a mass of 139.6 MeV/$c^2$, composed of up and antidown (u$d$) quarks and with a mean life time of $2.6 \cdot 10^{-8}$ seconds. The goal of the experiment, as already said, is to measure the branching ratio of the very rare $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay, where the branching ratio (BR) is the ratio between the number of particles decaying in a particular way and the total number of decaying particles of the same type. This decay has a remarkably clean nature from a theoretical point of view and can be reliably detected if a strong background suppression is guaranteed; all these properties makes this decay a decisive test for the SM. The predicted BR from SM is very small, $(7.80^{+0.85}_{-0.77}) \cdot 10^{-11}$, while until now the decay is very poorly measured, with only seven candidates events detected in the Brookhaven experiment. NA62 is therefore a very sensitive test, with the possibility of indirectly revealing the existence of new particles, and is complementary to direct searches at the LHC.

1.2 NA62 DETECTORS AND FEATURES

In order to measure the rare $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay and suppress the background particles, NA62 [4] has many detectors and features in the beam line, which can be seen in Fig. 1.4. From the left, protons from the 450 GeV hadron beam coming from the SPS hit a beryllium target and produce a secondary charged beam of 75 GeV in which around 6% of

![Fig. 1.4 - NA62 schematic showing the position of all the detectors of the experiment, installed in the ECN3 in the North Area Facility of the SPS [10].](image-url)
the particles are K\(^+\), while the others are \(\pi^+\) and protons. The identification of K\(^+\) is therefore important because \(\pi^+\) and protons can interact and be misinterpreted as signal events; this is achieved by the KTAG, that is a Cherenkov threshold detector (CEDAR). The GTK, or Gigatracker, belongs to the particles tracking system and is a kaon spectrometer comprised of three Si micro-pixel stations measuring time, momentum and direction of the beam particles. The other part of the tracking system is STRAW, that is a pion spectrometer consisting of four straw tube chamber measuring coordinates and momentum of secondary charged particles of the beam. STRAW is built of ultra-light material and is installed inside the vacuum tank in order to minimise multiple scattering. A Ring Imaging CHerenkov (RICH) detector allows separation between pions and muons and consists of a 17-meter long tank filled with Neon gas, acting as a radiator, at the pressure of one atmosphere; together with KTAG, it is one of the detectors of the particles identification system and is also useful to separate positrons from \(\pi^+\). Finally, the particles identification system is completed by a charged hodoscope (CHOD) in order to identify charged particles. Redundancy in muon identification and rejection comes from the MUon-Veto (MUV) system, consisting in two hadron calorimeters and a hodoscope. A photon-veto system is also required, in order to reach the requested background suppression; with respect to the beam line, the veto system must cover an angle region from 0 to 50 mrad from the beginning of the vacuum tanks. Because of this, it consists of four different types of calorimeters, or particle detectors which can track and measure energy of the crossing particles. First type of calorimeter is Large Angle Veto (LAV), that covers an angle interval from 8.5 to 50 mrad and is composed of twelve LAVs, positioned along the vacuum tanks of the experiment, in order to satisfy the required angle coverage. For very small angles, i.e. below 1 mrad, an Intermediate Ring Calorimeter (IRC) and a Small Angle Calorimeter (SAC) are used, while a Liquid-Kripton (LKr) electromagnetic calorimeter provides the photon-veto for the angles between 1 and 8.5 mrad. Moreover, LKr calorimeter identifies and rejects also muons, positrons and electrons, improving the particles suppression. Finally, an additional background suppression is supplied by CHANTI, consisting of scintillators surrounding the beam that vetoes charged and neutral particles. An important system is the Timing Trigger and Control (TTC) one, which optically distributes a common 40-MHz clock to all the experiment, together with Level 0 trigger information (see paragraph below).
Chapter 1. NA62 Experiment

1.3 Trigger and Data Acquisition System

The NA62 experiment consists of 12 detectors and several control systems, for a channel count of around a hundred thousand; only for the LKr calorimeter there are 13248 channels, continuously digitized with 40 MHz 14-bit ADCs, for a total data rate of 927.36 GB/s. A high-performance data acquisition system is therefore needed. In principle, a system in which all the detectors continuously send data to PCs, where they are analyzed and stored, would be the optimum in terms of reliability and data analysis, but would require a processing power and an amount of memory far too expensive to realise. In order to reduce the data rate that has to be analyzed and stored, a three-level trigger system has been implemented. The two systems, trigger and data acquisition, have been unified in the Trigger and Data Acquisition system (TDAQ), together with the TTC, looking for a high-performance system which can satisfy the rate requirements in a simple and cost-effective implementation.

Data rate is first significantly reduced by a hardware Level 0 (L0) trigger, based only on a few detectors; its algorithm will collect events with a single track in the CHOD, nothing in the MUV and no more than one cluster in the LKr. These detectors were chosen because CHOD can identify a charged particle within the acceptance of a K⁺, MUV can veto muon events by revealing the presence of muons and LKr can allow no more than a single cluster of particles compatible with a π⁺. It is possible to include other detectors in the L0 in order to refine it and to give more specific information to upper levels of trigger; RICH and STRAW can improve CHOD performance in charged particle identification, while LAV can enhance MUV and LKr photon and muon veto. The L0 algorithm is implemented in the L0 Trigger Processor (L0TP), which will time-match data received by the involved detectors, check if hypotheses are satisfied and generate a trigger signal.

During the L0TP evaluation, data from all the detectors will be stored in front-end buffers and if a positive L0 trigger is received the detectors will send the data to PCs, where upper-level triggers and data analysis algorithms are implemented. Otherwise, if no positive L0 trigger is received within a defined latency, the data stored in the buffers can be discarded.

Level 1 (L1) is a software trigger, based on independent conditions on single detectors; due to the impossibility of neutrinos detection, after a data quality verification L1 trigger looks for the presence of a single π⁺, in order to detect the wanted K⁺ decay. All L0 triggered events must be assessed by the L1 algorithm, and no data can be discarded until the L1 decision is taken. L2 trigger, again a software trigger, is based instead on
correlations between different detectors, so events have to be partially reconstructed to get a L2 decision; if the decision is positive, then, the event will be logged to tape, otherwise it will be deleted. Considering that one of the main topics of this work focuses on the use of TELDES project and that TELDES is part of the LKr calorimeter L0 trigger circuitry [11], a deeper analysis of the calorimeter is required. Each of the 13248 channel of the calorimeter consists of a 2cm-side cell of thin copper-beryllium ribbon; a Calorimeter REAdout Module (CREAM) provides amplification and 40 MHz 14-bit sampling to the channel, by means of a FPGA. Each CREAM board serves 32 calorimeter channels, buffering data that have to be sent to L1 trigger and summing 16 vertical cells (tiles) for the L0 circuitry and the two sums are sent to the trigger system by two of the twisted pairs of Cat.6 cable. The schematic of the system can be seen in Fig. 1.5. Calorimeter cells are grouped in slices parallel to the vertical axis each one containing 32 sums, for a total count of 864 sums. Also the LKr L0 trigger processor is a three-layer system, composed of Front-End (FE) and Concentrator boards, as in Fig. 1.6, both based on TEL62 boards.

![Diagram of LKr readout and trigger systems](image)

Fig. 1.5 - LKr readout and trigger systems, also showing a picture of the calorimeter before the installation.

Each of the twenty-eight FE boards receives a slice of the calorimeter by means of two TELDEEs, performs a peak search in space and computes time, position and energy for each detected peak; these computations are then sent to the second stage boards, with the precaution of sending a peak information to two boards for redundancy and to take boundary effects into account. The seven second-layer Concentrators horizontally merge different vertical peaks when they are close in time and space, in order to reconstruct clusters, each one corresponding to the interaction of a particle. The third stage consists in a Concentrator collecting all the information and sending data to the L0TP for the trigger decision, through four dedicated Gigabit Ethernet (GbE) links. One of the boards that have been tested in this thesis work, i.e. TELDEEs, were integrated during the 2014 test run,
where only a reduced system was installed, with six FE boards receiving data from the six central slices and a single Concentrator board only.

Fig. 1.6 - LKr L0 processor block diagram; GbE stands for Gigabit Ethernet.
Chapter 2. Hardware Environment

This thesis work consisted in the design of the test equipment for two boards, TELDES and InterTEL, developed by the INFN Perugia and Roma Tor Vergata sections. In order to reach a better comprehension of the choices made for such equipment, a description of the two projects is mandatory.

The main components of TELDES and InterTEL projects are Texas Instruments (TI) DS92LV16, DS15EA101 and DS15BA101 chipsets, used in the digital communication systems of the LKr calorimeter and other detectors. The firmware developed for the test equipment has been implemented on an Altera Cyclone IV FPGA, included in a Terasic DE2-115 Development Board. Both TELDES and InterTEL are daughter cards for the TEL62 boards, so a concise description of its architecture is given.

2.1 DS92LV16 Serializer/Deserializer Chip

The DS92LV16 [12] is a chip that implements Physical and Datalink layers of the ISO/OSI stack, containing a serializer and a deserializer, as in Fig. 2.1, that can serialize 16-bit LVCMOS/LVTTL input data to a Bus Low-Voltage Differential Signaling (BLVDS) and vice versa, representing an alternative for 8b/10b chipsets.
As it can be seen, the device contains two separate blocks for serializer and deserializer, which can operate independently from each other, with separate power-up and power-down controls. The chip can then be used for low-power applications, when only serialization or deserialization is required, as in the case of TELDES project. DS92LV16 supports clock frequency in the 25-80 MHz range, implying a maximum data rate of 1.28 Gbps for a single block or 2.56 Gbps for Full Duplex applications.

Datalink layer is implemented with a simple algorithm, that is framing the sixteen serialized bit with two embedded clock bits, the first always high and the last always low (see Fig. 2.2); these bits produce a rising edge transition providing a precise timing information for each data word. With such embedded clock, the receiver PLL can also lock to random data, eliminating the need for the transmission of training patterns, which is mandatory in other protocols.

DS92LV16 [13] has three operating states, i.e. Initialization, Data Transfer and Resynchronization, in addition to two passive states that are Powerdown and TRI-state. Each state has a different behaviour, as described in the following list, referring to a point-to-point architecture:

- Initialization: before sending data, a DS92LV16 has to initialize the links to and from another DS92LV16. The state is divided in two additional steps: first, the outputs are held in tri-state while \( V_{CC} \) reaches a suitable level (2.2 V). Afterwards, both the serializer and deserializer PLLs have to lock to local clocks, called TCLK and REFCLK respectively; their period ratio must be in the 0.95-1.05 range, i.e. REFCLK frequency has to be ±5% of TCLK frequency, in order to let the deserializer work properly. During these operations all the outputs are held in tri-state, while the deserializer /LOCK output is maintained high. In the second step, the serializer sends the LVDS data stream while the deserializer PLL has to synchronize to it. Data stream can be either random non repetitive data or SYNC sequence, i.e. "1111111000000000" words. In order to lock to the incoming signal, the deserializer has to identify the low-high transitions of the embedded
clock: this process takes a specified amount of time, that is a hundred and fifty clock cycles if a SYNC pattern is being transmitted, or an unknown number of clock cycles over random data. These data have to be non repetitive because if there are two consecutive bits forming a low-high transition during multiple clock cycles (Repetitive Multiple Transitions, RMT) they can be interpreted as the embedded clock by the deserializer, that will not lock until the change of these bits takes effect. Once PLL lock to LVDS incoming data, /LOCK pin goes low and valid data appears on the output.

- Data Transfer: after the Initialization, the serializer effectively sends valid data to the deserializer, which produces a 16-bit data output and a RCLK output, containing the clock recovered from the incoming LVDS signal. Data are valid until the /LOCK pin is low, otherwise are invalid and not to be considered.

- Resynchronization: the deserializer PLL tries to resynchronize to the LVDS signal once the lock is lost; losses of lock are possible after two consecutive misdetections of the embedded transition, while they are not affected by RMTs. To indicate that the output data are not valid, the deserializer /LOCK is driven high as long as it remains in the Resynchronization state. In order to relock on data stream, the deserializer will behave exactly as in the second step of the Initialization state; the serializer SYNC port and the deserializer /LOCK port can be connected so that a SYNC pattern can be sent if a loss of lock occurs.

- Powerdown: this is a low-power state that the serializer and deserializer enter while waiting for Initialization or when the TPWDN and RPWDN ports are driven low. PLLs are stopped and outputs are held in tri-state, causing a reduction of the consumed current, that reaches the order of µA. The device automatically enters the Initialization state when it exits from Powerdown.

- TRI-state: when REN pin is driven low, the deserializer enters TRI-state by holding the output data and RCLK in tri-state; whenever the REN pin returns high, the deserializer will return to the previous state. Also the serializer has the TRI-state state, which is controlled by the DEN pin and affects the LVDS output.

For the Physical layer of the ISO/OSI stack, DS92LV16 implements the ANSI TIA/EIA-644-A-2001 standard [14], also known as LVDS [15]; a brief description of it is then required. A typical LVDS interface circuit is shown in Fig. 2.3 and consists of a transmitter, a balanced transmission line and a load part.

![Typical LVDS interface](image)

Starting from the last one, the load is composed of the receiver itself and a termination resistance, the impedance of which depends on the characteristic impedance of the balanced transmission line; typical values are 100-120 Ω, with a maximum tolerance of ±10%. The transmitter, instead, consists of a current generator that injects a constant current, typically 3.5 mA, in the transmission line; most of the current will flow through the termination impedance, causing a voltage difference across it. For a 100 Ω resistor and 3.5 mA current, the voltage difference will be 350 mV, while the minimum differential input voltage established by the standard is ±100 mV. The information on the logic level is in the polarity of this voltage, depending on the direction of the injected current. An offset voltage is maintained with a typical value of 1.25 V. The Maximum data rate is not defined in the standard, although a theoretical maximum limit is calculated at 1.93 Gbps assuming a lossless transmission line. The EIA-644-A-2001 standard defines point-to-point and multi-drop architectures, widening the number of possible applications of it. When referred to multi-drop architectures, LVDS assumes the name Bus LVDS, or BLVDS, and differs from point-to-point LVDS in providing increased drive current in order to handle a greater number of receivers and terminations.

The benefits of using LVDS signals instead of single-ended signals mainly derive from the differential architecture: electromagnetic noise will equally affect the two wires of the transmission line, appearing as a common mode noise, which does not affect the differential voltage. In addition, the constant current consumption simplifies power supply, reducing the interference in the ground and power supply lines. Because LVDS reaches
data rate on the order of Gbps even using twisted-pair copper cables, due to the robustness of the transmitted signals, it is one of the most popular data transmission standards and is used in many high-performance applications.

### 2.2 DS15BA101/DS15EA101 CABLE EXTENDER CHIPSET

![Diagram of DS15BA101 and DS15EA101 chipset](image)

**Fig. 2.4** - TI recommended settings for DS15BA101 and DS15EA101 as cable extender chipset; images of a hypothetical signal show its evolution over the system.

DS15BA101 [16] and DS15EA101 [17] are two Texas Instruments chips used together as high-speed differential cable extender chipset, as shown in Fig. 2.4; they were included in the TELDESs and InterTELs boards in order to allow a reliable communication also with long cables, facing the noise coming from the harsh environment of the experimental area. DS15BA101 is a cable driver and signal buffer used both in differential and single-ended applications, reaching data rates up to 1.5 Gbps with low power consumption (150 mW typical). It has two output levels, controlled by the value of an external resistor, $R_{VO}$: for differential signals, output is 800 mV$_{pp}$ with a $R_{VO}$ of 953 $\Omega$, while for single-ended signals is 800 mV with a $R_{VO}$ of 487 $\Omega$. DS15EA101 is an adaptive equalizer, once again both for single-ended or differential signals; it can manage a maximum rate of 1.5 Gbps or a maximum cable attenuation of 35 dB at 750 MHz. The input level has two possible values, whether the signal is differential or not: in the first case, input has to be 800 mV$_{pp}$, while it has to be 800 mV in the second one. As already said, the DS15EA101 together with the DS15BA101 forms a cable extender chipset, and is optimized for extending serial data streams from serializer/deserializer pairs over 100 $\Omega$. 
differential cables; Fig. 2.4 shows TI recommended settings for the purpose. The two possible launch amplitudes of the DS15BA101 are chosen because are the ones that minimize the jitter amount of equalized signals for differential and single-ended cables, as can be seen in [18]. DS15EA101 embeds an output port, named LOS, that points out if a valid signal is present or not: when a DS15EA101 can sense a valid input signal, the port is driven low, while if the signal is lost it is driven high. Although any communication link based on LVDS standard and with a rate included in the 0.15-1.5 Gbps range may benefit from this cable extender chipset, serializer and deserializer chipsets that operate with a single DC-balanced data stream with embedded clock are best suited for it. Because of this, the DS15BA101/DS15EA101 pair has been chosen for the coupling with two DS92LV16 in the communication link of the LKr electromagnetic calorimeter.

2.3 Field-Programmable Gate Array

A Field-Programmable Gate Array (FPGA) [19] [20] is an integrated circuit that consists of an array of logic cells designed to be configured after manufacturing by the customer, i.e. field-programmable. Even if this definition is still valid, nowadays FPGAs have more than simple logic cells: memory blocks, such as RAM blocks, are always included, together with Digital Signal Processing (DSP) cells, usually multipliers, in order to speed up simple common operations and to reduce the required area. More recently, also hardware processors and microcontrollers are included, with the purpose of increasing the flexibility of the circuit. Historically, FPGAs were slower and less efficient than the Application Specific Integrated Circuit (ASIC) alternative, but progressively these differences are diminishing; furthermore, FPGAs are re-programmable devices, then with the possibility to be updated in any time, even after the sale. These properties lead to shorter time-to-market and higher flexibility than ASICs, so that vendors tend to prefer FPGAs over ASICs. In those applications that require the highest performance with the lowest power consumption, ASICs are preferred, but FPGAs can still be used for the prototyping process before the final ASICs production. FPGAs are also valid alternatives for the computer-based solutions because they can implement hardware parallelization by reserving part of the logic cells for a dedicated task, instead of sharing CPU and memory resources, even if this is counterbalanced by a higher development complexity. Configuration and behaviour definition of a FPGA are generally specified by a Hardware Description Language (HDL), which can be Verilog or VHDL, even though circuit diagrams and other graphical tools can be used, e.g. Finite State Machine (FSM) editor.
The configuration set is called firmware and a deeper analysis of HDL firmware description is provided in Chapter 3. The most common architecture of a FPGA consists of logic blocks, Input/Output (I/O) blocks and an interconnection matrix, as can be seen in Fig. 2.5. Depending on vendors, the logic block is called Configurable Logic Blocks or Logic Array Block (LAB); from now on, logic blocks will be called LABs, because FPGAs used are Altera devices. Each LAB has a few sub-elements, or logic cells, that are called Logic Elements (LE) according to the Altera nomenclature system. The simplest architecture of a LE is shown in Fig. 2.6: the 4-input Look-Up Table (LUT) is used in defining cells behaviour, implementing a Boolean function that returns a single output depending on the input combinations. D-type flip-flop registers LUT output and multiplexer decides whether the LE output has to be registered or not. LE are today far more complex than the one described above, including many other devices, but they maintain the same basic behaviour.

![Diagram of FPGA architecture](image)

Fig. 2.5 - Basic architecture for a FPGA, showing the elementary components.

One of the most common features included in LEs is a dedicated Full-Adder (FA), which improves performance and minimizes the area consumption of a design, also reducing its development time. In order to include the FA, some changes are needed in the LE architecture, with the introduction of two dedicated multiplexers, as shown in Fig. 2.6; a real example of a LE architecture can be found in paragraph 2.4.1, where Cyclone IV FPGA is described. I/O blocks manage signals to and from FPGA and are placed near its
sides; they can be both input or output and have to include at least an input and an output buffer in order to guarantee signals integrity. Depending on FPGA, then, other features are also available, such as delay lines or input/output registers. Fig. 2.11 in paragraph 2.4.1 2.4 shows an I/O block of a Cyclone IV device.

![Fig. 2.6 - Two possible architectures for a simple LE, either with or without a Full-Adder.](image)

Finally, the interconnection matrix connects all the FPGA blocks, improving the complexity of functions that can be implemented if two or more LABs are connected together, linking separated features of FPGA and connecting I/O with internal blocks, always minimizing time delay and signal distortion. It is composed of unmovable lines for signal transmission and of switch matrices, i.e. programmable pass-transistor networks that have routing features.

As already said above, besides this basic architecture there are many other components included in FPGAs, such as multipliers, basic DSP blocks, memories and high-speed I/O blocks. Also microcontrollers, Ethernet and external memory controllers are becoming more common, and it is worth noting that also analogue circuitry is included in FPGAs, in order to implement features which cannot be built out of LUTs, e.g. transceivers for specific communication protocols such as Ethernet ones. Finally, some circuitry for clock generation and management is always included: a very important feature is the Phase-Locked Loop (PLL) which can generate multiple clock outputs with defined frequency and phase from a single clock input; this lets consumers synthesize new clock frequencies from a single clock input, usually with lower jitter amount. Because FPGAs have a lot of synchronous circuitry, a dedicated clock distribution network delivers clock to all FPGA blocks with the smallest skew possible.

### 2.4 DE-2 115 DEVELOPMENT BOARD

For the FPGA implementation of the test equipment a *Terasic* DE2-115 Development Board [21] [22] has been used, which is shown in Fig. 2.7. The most important component of the board is the *Altera* Cyclone IV E FPGA, which controls all the
other DE2-115 features. The following is a list of some components included in the board:

- 18 switches and 4 push-buttons
- 26 user LEDs and 8 seven-segment displays
- 1 LCD module
- 2 Ethernet ports with relative controllers and tranceivers
- USB I/O controllers
- Audio/Video I/Os
- 50 MHz clock oscillator
- SMA external clock I/O
- 2x64 MB SDRAM, 2 MB SRAM and 8 MB Flash memories
- 40-pin expansion connector
- QTS expansion connector

DE2-115 is then a high-end educational development board, offering a rich amount of interfaces in order to guarantee versatility and usability in many fields, such as image and video, networking and digital signal processing, always paying attention to low cost and low power consumption.

Components used in the test equipment, except for FPGA, are the 40-bit expansion header, the SMA external clock input, the LEDs and the switches and buttons; also LCD
module and seven-segment displays were used in a former version of the firmware, but were removed since they are no more useful when the board is used remotely.

### 2.4.1 Altera Cyclone IV E FPGA

Cyclone IV [23] is a 60-nm FPGA family with E or GX variant, depending on the presence or not of embedded transceivers; the E variant does not have that transceivers and comes in nine different models, each one with a different number of included components. The DE2-115 board embeds a EP4CE115 Cyclone IV model, that is the best FPGA of the E series, featuring about 115 thousand of LEs, 3.9 MB of memory, more than two hundred 18-bit multipliers and up to 4 general purpose PLLs. A view of the whole FPGA is shown in Fig. 2.8.

![Fig. 2.8 - View of the Cyclone IV EP4CE115 FPGA; blue blocks are LABs, white blocks are DSPs, green are memory blocks and brown are I/O blocks. In black, empty blocks are represented.](image)

Each Logic Array Block, shown in Fig. 2.9, consists of sixteen Logic Elements, connected to each other by local interconnect, organized in rows and columns, which is then linked to the global interconnection matrix. Also neighbouring blocks of any type, i.e. LABs, memories or DSPs, can drive the local interconnect of a LAB through a direct link connection, in order to provide higher performance and to minimize the use of the interconnection matrix. Each LE can drive up to forty-eight LEs through local and direct link. Carry and Register chains are two connections between adjacent LABs that transfer
carry and LE outputs; a network is dedicated to control signals, together with the proper logic, and distributes two clock signals, two clock enable, two asynchronous clear, one synchronous clear and one synchronous load. Each LE can be connected to one of the two clock signals, using the relative clock enable, while clear and load signals affects all the LEs of the LAB.

Fig. 2.9 - Cyclone IV LAB architecture, including 16 LEs, local interconnects and direct link connection.

Architecture of Cyclone IV E LEs is more complex than the one of the general model described in previous chapters, and is reported in Fig. 2.10. As it can be seen, the basic structure, i.e. a four-input LUT and a flip-flop, is the same as the general model, but the flip-flop can be configured as D, T, JK or SR.

Fig. 2.10 - Block diagram of a Cyclone IV Logic Element.
Each LE has three outputs driving column, row and local interconnection, and can be used independently by LUT and flip-flop in order to improve LE utilization. Finally, a register feedback network is worth noting, together with carry and register-chain outputs that are connected to LAB carry and register chains. LEs have two operating modes, normal and arithmetic; the first is suitable for general logic applications, either synchronous or asynchronous, and supports register feedback network, while the second is optimized for adders, counters, accumulators and comparators, implementing 2-bit FA and using the carry chain.

I/O blocks feature many assets, whether they are configured as input or output blocks: the basic version implements only an input or output buffer and a connection to external pad; output flip-flop, either with an output-enable port or not, is available for output blocks, while an input flip-flop is included in input ones. Finally, some delay lines are placed upstream the output buffer and downstream the input one. A view of an I/O block is available in Fig. 2.11.

![Fig. 2.11 - Block diagram for a Cyclone IV I/O block; the block shown is configured as an output block, and flip-flop, output buffer and pad are highlighted in blue.](image-url)
2.5 TEL62 BOARDS

TEL62 [24] is a general-purpose board developed by the INFN Pisa section for the TDAQ system and used in all NA62 detectors; it is an upgrade from the TELL1 board for the LHCb experiment at CERN, maintaining the same overall architecture, but embedding more powerful devices in order to handle both data and L0 trigger flows of the experiment. TEL62 is a 16-layers Printed Circuit Board, with all lines controlled in impedance (50 Ω) and complying with the 9U Eurocard standard; it is used in powered VME crates, which provide different supply voltages and 32-bit address and data buses. The main components of TEL62 are *Altera* Stratix III L [25], which are 65-nm high-end FPGAs. Although there are many differences between Cyclone IV and Stratix III, the basic structure is the same: LABs, I/O blocks and interconnection matrix are the basic components, even though they consist of completely different circuitries. Each LAB includes ten Adaptive Logic Modules (ALMs), the basic logic cells corresponding to Cyclone IV LEs, and all the connections as in Cyclone IV, i.e. local interconnect, direct link, control signals, carry chains, register chains and arithmetic chains. What really differs from Cyclone IV architecture are the basic logic cells: each ALM consists of two six-input LUTs combined with two dedicated FAs, two flip-flops and all the circuitry needed for interconnections and controls. ALMs can operate in five different modes: Normal mode and Arithmetic mode are the same as Cyclone IV, and are suitable for general logic applications or arithmetical applications respectively.

![Fig. 2.12 - Block diagram of a Stratix III Adaptive Logic Module, showing two six-input LUTs, two FAs and two registers.](image-url)
While in Normal mode an ALM can implement up to two six-input functions, in the Extended LUT mode each cell can implement a specific set of seven-input functions, e.g. a 2-to-1 multiplexer; these functions often appear if a conditional statement is used in the HDL description of the system. An extension of the Arithmetic mode is the Shared Arithmetic mode, in which each ALM can behave as a three-input adder. The last mode is the LUT-Register one, which allows third register capability within the ALM. A high-level view of an ALM of the Stratix III L FPGA is shown in Fig. 2.12.

TEL62 board has five Stratix III FPGAs: four of them, called Pre-Processing (PP) FPGAs, are connected to a daughter card through a QTS connector and to a 2-GB DDR2 memory buffer; each PP is then connected to a central FPGA called SyncLink (SL) through two independent 32-bit data buses. A daughter card (AUX) and a Quad Data Rate (QDR) SRAM completes the SL connection, together with an output board. A TTCrx board decodes clock and L0 trigger information from the TTC system, while a commercial Credit-Card PC (CCPC) running Linux provides control, monitoring and configuration to the board, together with a custom I/O interface card (GLUE) that implements JTAG, I2C and ECS protocols.

![Fig. 2.13 - TEL62 [26] board used for the LKr calorimeter, with two TELDE5s plugged on four QTS connectors.](image)

Data and L0 trigger information have two different flows: data flow starts with PPs receiving data from daughter boards and, if no errors are detected, continues with their
storage in the DDR2 buffers, organized in time slots of 25 ns. A counter is incremented every time a word is written in a time slot, in order to time-stamp each stored word. When a L0 trigger reaches the TEL62 through the TTCrx board, it is time stamped, so the corresponding time slots are sent from PPs to SL, where they are merged and sent to dedicated PCs through the output board. Also the L0 trigger flow starts with PPs receiving data from daughter boards, but data, then, are processed and sent to SL, which sends the information to the L0TP; the processing algorithm depends on the detector where the TEL62 is used.

Fig. 2.13 shows a TEL62 used in the LKr L0 trigger circuitry and, in the left, two TELDES boards are connected as daughter cards for the four PPs.

2.6 TELDES PROJECT

![Fig. 2.14 - From the left, top and bottom view of a TELDES [26] board are shown: in the top, sixteen equalizers and deserializer can be seen, while in the bottom the Ethernet and QTS connectors.](image)

TEL62 DESerializers, or TELDESs, are seventy 8-layer boards on FR4 substrate, developed by the INFN Perugia and Roma Tor Vergata groups for the L0 trigger system of the LKr electromagnetic calorimeter and one of them is shown in Fig. 2.14.

Each TELDES connects eight CREAMs to two PP FPGAs of a TEL62 by means of eight RJ-45 connectors; indeed, each CREAM sums two tiles of the calorimeter and serializes the resulting 16-bit words, in order to transfer it to the FE TEL62 of the L0 circuitry, and more specifically to two channels of a TELDES. Because the physical link between a CREAM and two TELDES channels consists of only two twisted-pairs, sums from the CREAMs have to be serialized and consequently deserialized on the TELDESs;
this operation of serialization/deserialization (SERDES) is accomplished using two TI DS92LV16s, one as a serializer and one as a deserializer. Originally, a CREAM had to be connected to two TELDES channels by two of the four pairs of a Cat. 5e [27] Unshielded Twisted Pair (UTP) 15-meters-long cable via a RJ-45 connector. From [18], each SERDES has distance limitations that depend on many factors, i.e. the pattern characteristic, reference clock, power and system noise, connectors and, the dominant one, cables. Depending on the cable, the characteristics that affect the link are attenuation and crosstalk; a Cat. 5e cable attenuation, or insertion loss, is reported on Fig. 2.15.

![Fig. 2.15 - Cat. 5e insertion loss per meter as a function of the frequency [18]; 360 MHz is highlighted.](image)

Due to a data rate of 720 Mbps, the signals can be considered as sampled at 720 MHz, i.e. with a bandwidth of 360 MHz. The insertion loss at this frequency is about 0.45 dB/m, meaning a total attenuation of 6.75 dB for a 15-meter cable. For the remaining part of this paragraph, every figure refers to the DS92LV18 SERDES devices, which have the same behaviour of the DS92LV16 but deserialize an 18-bit word to a LVDS data stream with embedded clock. From extensive empirical data collecting, Texas Instruments have shown that its SERDES devices can handle a total cable loss between 3 dB and 6 dB when considering the total raw data rate. Because of this, DS15EA101 was installed in TELDES boards, together with DS15BA101 into CREAMs. Fig. 2.16 shows the achievable
distances when the DS15BA101/DS15EA101 pair is used: a maximum Cat. 5e cable length of around 85 meters is achieved for a rate of 720 Mbps; this graph is inferred by extensive empirical data collecting.

![Graph showing maximum rate vs cable length](image)

Fig. 2.16 - Typical performance for DS92LV18 over Cat. 5e, with the cable extender chipset [18].

Two issues showed up during the installation in the experimental area of the NA62 experiment, which have been solved thanks to the collaboration with the INFN Tor Vergata group: firstly, a DS15EA101 could sense the data stream of the adjacent twisted pair, highlighting a cross-talk issue, and secondly the system suffered environmental Electromagnetic Interference (EMI). Because of these, the DS15EA101 drove high its LOS port, pointing out that there was no valid signal at its input port and providing corrupt signals to the deserializer, which drove high its /LOCK port too. Together with the Tor Vergata group, a check of the two signals was then added to the PP FPGAs of the TEL62, in order to count how many times the noise significantly interfered with the system.

![Diagram of Cat. 6 cable section](image)

Fig. 2.17 - Cat. 6 cable section.
Both the two issues were solved using Shielded-Foiled Twisted Pair (SFTP) Cat. 6 [28] cables: as it can be seen in Fig. 2.17, each pair is shielded with a metallic foil that reduces cross-talk while external jacket and braid provide additional isolation from the external EMI.

2.7 InterTEL Project

InterTELS are so called because they interconnect two TEL62s by means of a LVDS link; on a FR-4 substrate, these 4-layer boards are shown in Fig. 2.18, and include:

- Two 40-MHz oscillators connected to two clock buffers,
- Two DS92LV16 SERDES,
- A DS15BA101 and a DS15EA101,
- Two RJ-45 connectors,
- Two signal buffers,
- A 60-pin fine pitch SMD connector.

![Fig. 2.18 - From the left, top and bottom view of an InterTEL [26] board can be seen: the top includes two DS92LV16s, a DS15EA101 and a DS15EA101, together with signal buffers and clock devices; the bottom shows the Ethernet and 60-pin fine pitch SMD connectors.](image)

They have been developed by the Electronic Service of INFN Perugia for CEDAR, RICH and LAV detectors, for a total count of 22 boards, in order to interconnect two or more of
their TEL62 for information sharing and particles detection enhancing. In RICH and LAV, then, InterTELs are also used to refine their L0 trigger, improving precision and decreasing the number of false-positive triggers. Each TEL62 is connected to the other by using the SERDES chipset of the InterTEL, although the detectors still have not chosen the topologies; together with the working groups of the three detectors, two possible configurations have been foreseen, that are daisy-chain or star topologies, each one with different strengths and weaknesses. With the first one, two InterTELs are at a maximum distance of 10 meters away, but the SL FPGA of the last TEL62 has to handle a large amount of data, causing possible reliability problems; the second topology foresees an additional TEL62 dedicated to this task, that unloads the SLs processing power constraints but requires cable that are far longer than in the other case. The choice of the topology will be taken in the next months by the INFN Perugia section together with the detectors working groups. InterTEL boards are plugged to the TEL62 with the 60-pin connector, connecting the SL FPGAs with both control and information signals; because of space constraints when plugged, InterTELs have the form-factor shown in Fig. 2.18: all signals to and from the SL have to pass through the bottleneck near the 60-pin connector, causing interference and noise issues. In order to reduce these issues, the two signal buffers were included in the boards, one for information and one for control signals.

One of the two DS92LV16s is used as a serializer, generating a LVDS data stream from 16-bit words of a TEL62, while the other is used as a deserializer, receiving the LVDS stream of a TEL62 and sending 16-bit words to the board onto which it is plugged. Although each DS92LV16 has separate serializer and deserializer blocks, a double architecture has been preferred in order to have two actually separated blocks for the transmitter and the receiver. Because of the space constraints, however, a possible improvement may be the use of a single DS92LV16 both as serializer and deserializer, which would also allow to remove one of the oscillators and one of the clock buffers. The reduction of elements, then, will also lower power consumption and the possibility of components failures, although this is not required in the environment. The link between two InterTELs consists of a Cat. 6 cable, exactly as in the TELDES case, but with different lengths, that can vary from ten to eighty meters depending on the detectors and topology. DS92LV16 alone, then, can not face the cable loss, so the cable extender chipset formed by the DS15BA101/DS15EA101 pair has been included in the boards. For 720 Mbps signals, the chipset can reach distances around 85 meters, as it can be seen in Fig. 2.16, which is higher than the maximum cable length.
Chapter 3. Software Environment

Programming a FPGA means generating and loading a binary file that contains the information on how each logic block and each interconnection resource has to be configured. This is a very complex process, that starts with the description of the system with HDLs or other graphical tools, while a dedicated design suite implements algorithms for transformation and optimization of the firmware in order to suite the employed FPGA. Due to the complexity of such algorithms, a design flow containing both design and verification steps guides the programmer through the complete development process, in order to guarantee the efficiency and the efficacy of the resulting firmware.

Quartus II environment has been used for FPGAs design, implementation and programming, while behavioural simulations of the system have been carried out with ModelSim Altera-edition software. Other tools of the Quartus II suite have been used for in-circuit testing and for remote control of the FPGAs.

The VLSI Hardware Description Language (VHDL) has been preferred over the block editor for the design of the test equipment, mainly because it is more powerful and flexible and because it allows an easier management of connections between blocks. Although Verilog HDL would have the same benefits, the choice of VHDL comes from uniformity reasons: TEL62 FPGAs includes designs mainly described in VHDL and, in the former version of the firmware, data from Cyclone IV FPGA were sent by a Xilinx proprietary module written in VHDL. Because VHDL has been used, a brief description of it is given in order to reach a better comprehension of the next chapters; complete lectures about this topic can be found in [29] [30].

3.1 FPGA Design Flow and Quartus II environment

Altera Quartus II [31] environment leads the whole design flow, which is shown in Fig. 3.1, specifying both design and simulation steps. The first one is the Design Entry, in
which all the source files, i.e. files needed for FPGA implementation of the system, are created and included in the project; design files contain the description of the system and are organized in hierarchical modules, with a Top Module as the most important one. The description of a module in a design file can be carried out using different tools from Quartus II: a block editor is the simplest one, with graphical inclusion and connection of the building blocks; when system complexity increases, the block editor becomes uncomfortable due to the large number of components and connections that have to be included, and is replaced by the text editor, which describes the system by means of hardware description languages. The State Machine Editor is a graphical tool for state machine design, in which the user can define states and variables and can specify the behaviour of the machine through the definition of the transitions. Finally, with the Mega Wizard Plug-in Manager, Intellectual Property (IP) functions, also called MegaFunctions, can be included in the design. In addition to design files, Quartus II Settings Files (.qsf) contain assignments and settings of the project; based on Tcl script syntax, they specify the properties of target FPGA, I/O pins used, connections between physical FPGA pins and

![Fig. 3.1 - FPGA design flow.](image-url)
project virtual names, standards that pins have to comply with and configuration of some FPGA blocks, like the I/O ones. These settings could be edited through two Quartus II tools, i.e. Assignment Editor and Pin Planner, or through a simple text editor. A complete description of Tcl script syntax would be beyond the aim of this work, so only commands used for the project will be reported. In order to check the logical behaviour of each module, a Behavioural Simulation is then recommended. This is carried out through a simulator software, which needs design files and a testbench file: the first ones describe the device that has to be tested (Device Under Test, DUT), while the second provides input signals to it. The simulator, then, interprets the DUT as a circuit and connects the stimuli included in the testbench to the relative input ports, providing logical results both for internal signals or output ports. The software used for simulations is the external tool Mentor Graphic ModelSim - Altera Edition [32].

Once the system has been designed and simulated, the Synthesis step builds a single project database that integrates all the design files in a design entity, used by Quartus II to examine the consistency of the project and to check syntax errors. The project is then synthesized and mapped to the specific FPGA, inferring flip-flops, latches and FSMs from the design files; the whole process searches for system optimization by removing redundant logic, minimizing gate count and creating assignments for FSMs, in order to use the FPGA architecture as efficiently as possible. The optimization algorithms can be customized depending on what has to be optimized, i.e. timing requirements or area and power consumption, with the possibility of a fine customization by Quartus II logic options, e.g. whether redundant logic has to be merged or not. Synthesis is followed by the Implementation step, also known as Place&Route, in which Quartus II matches the logic and timing requirements of the project to the available resources of the FPGA and attempts to match the assignments specified in the .qsf file. Three verification steps are parallel to Implementation: in the Static Timing Analysis step, TimeQuest Timing Analyzer verifies if timing constraints included in a Synopsys Design Constraint (.sdc) file are respected or not, returning an error in the second case; for example, .sdc files can include constraints over clock period, maximum delay from LE to LE or minimum delay from the input and output of a signal. Functional Simulation differs from Behavioural Simulation in considering synthesis and mapping of the project to the FPGA used, while Timing Simulation uses the information coming from Place&Route. The last simulation step shows the behaviour of the actual implementation of the project, taking into account the real delays between LEs and I/O pins. If the design passes the timing analysis with no errors, a
binary programming file (SRAM Object File, .sof) is generated and contains the configuration of all the FPGA LEs. In Device Programming, this file is uploaded to the FPGA by the Programmer tool through the JTAG standard and the device has now the behaviour specified by the designer; an additional verification step checks the validity of the design once it has been implemented in the real device and concludes the design flow. For the real-time testing of the design, Quartus II environment provides SignalTap II Logic Analyzer [33] tool. In addition to this, Quartus II includes also two tools for remote control of FPGAs, named In-System Memory Content Editor and In-System Source and Probes Editor, which allow to read from and write to running devices using JTAG; the former is used for in-system memories, while the latter for generic signals. The two tools can be used through two dedicated panes after the instantiation of \texttt{altsycram} or \texttt{altsource_probe} MegaFunctions in the design.

\section*{3.2 VHDL PRINCIPLES}

VHDL [34] [29] [30] is not a programming language, but a language used for electronic circuits description: this means that its statements describe elements of a system instead parts of an algorithm that an executor has to follow in order to reach a specific purpose. The language allows circuit description in two abstraction levels:

- **Gate level:** also known as Structural level, is the lowest abstraction level, where a circuit is described in logic elements, such as logic ports, multiplexers and flip-flops, and connection between them. The gate level representation of a design is called netlist, and it is the closest model to the real circuit implementation.

- **Behavioural level:** it allows to describe a system by one or more algorithms, while structure and transformations of the design are automatically carried out by the software suite during the Synthesis step. The design is divided into two kinds of elements: asynchronous circuits, which describes data transformations by equations and conditions, and registers, that store intermediate elaboration results.

Actually, in a VHDL project different modules can be described in different abstraction levels, or different abstraction levels can be mixed in the same module.

A VHDL design is organized in modules, i.e. sub-circuits dedicated to a specific task, connected together in a hierarchical structure, in order to realize more complex functions. As already said, the first hierarchical level is named Top Module, and its I/O ports are the overall I/Os of the project. In VHDL, a module has the name of Design Entity and is composed of two parts: the entity declaration and the entity architecture. The first
one specifies name, ports and parameters, called generics, of the Design Entity, with the syntax shows in the code below

```vhdl
entity entity_name is
    [generic(generic_list);] --parameters for the design entity
    [port(port_list);]     --elements of the list are separated by commas
end entity_name;
```

where "--" indicates a comment, i.e. all the text after the symbol will not be considered by the compiler but will only contain indications for readers. Ports have to be one of in, out or inout direction, and must be of a specific type, like each variable, constant and signal of the design. VHDL includes a large number of types, although only a few of them can be synthesized: one of them is the bit type, which represents a logical value and can assume only 1 or 0 values. Because it can not assume high-impedance or don't-care conditions, the bit type is not often used and is replaced by the std_logic one, included in the IEEE library std_logic_1164, which extends the number of possible states to nine: in addition to 0 and 1, there are don't-care, high-impedance, undetermined, undefined, weak unknown, weak high and weak low. In the std_logic_1164 library there is also the std_logic_vector type, which represents vectors of std_logic objects and is useful for bus descriptions. Another type that is worth noting is the integer one, which represents 32-bit integer numbers; VHDL allows designers to declare their own types by defining all the possible values that they can assume. Many other types are included in VHDL, but a deeper analysis of them goes beyond the aim of this thesis. Entity architecture contains the structure of the Design Entity, specifying its behaviour; the following syntax has to be respected:

```vhdl
architecture architecture_name of entity_name is
    [declarations]
    begin
        [implementation]
    end architecture_name;
```

Each architecture is associated to only one entity, but an entity can have multiple architectures, even though this is used only on very complex designs. In the declarations part, constants, user-defined types and signals can be declared, together with all the modules that have to be instantiated in the Design Entity. The declaration of a module is carried out by the component statement, specifying its generics and ports, with the following syntax:

```vhdl
component component_name is
```
The *implementation* part, instead, specifies the behaviour of the Design Entity containing instantiation of components, concurrent or sequential logic. Once declared, a component can be instantiated multiple times, as indicated below:

```vhdl
Instance_name : component_name
generic map (generic_association_list)
port map(formal_name => actual_name);
```

Each *formal_name* represents a port which is mapped to a signal or an I/O port of the upper entity, here called *actual_name*; there is also another way to map the component ports to external signals, that is writing all the *actual_names* in the same order in which the ports are declared in the component. The first method is called *association by name*, while the second *positional association*, and the choice between them is up to the designer. For what concerns concurrent logic, VHDL supports any kind of logical expression that contains basic operators, i.e. *not*, *and*, *or*, *xor*, and LUTs implementation by the *when* command. Operators for left or right logical shift are *sll* and *srl*, while *rol* and *ror* are command for circular shifts; also arithmetical operators such as addition and subtraction can be used in the asynchronous logic between *std_logic* and *integer* types either, while multiplication, division and exponentiation are only available for *integer* objects. Sequential logic is described inside one or more *process* statements, in accord to the following syntax:

```vhdl
[process_name]: process (sensitivity_list)
[declarations]
begin
[body]
end process;
```

The main difference from concurrent and sequential statements is that they are concurrently or sequentially interpreted by the software, as their name suggests: outside the *process* commands all the statements are executed in parallel, while inside they are executed in the order that they are written. Even if it contains sequential logic, *process* is a concurrent statement, and therefore is executed at the same time as the other concurrent statements. When a signal in the *sensitivity list* of a process changes, it activates that process and the statements of its *body* part will be executed; while concurrent logic can
only be asynchronous, sequential statements can be used to implement synchronous logic, simply by including a clock signal in the sensitivity list: with the rising_edge command, at every rising edge of the clock signal the commands included in the body part are executed. Some of the sequential statements are similar to the concurrent ones, for example the ones for logical expression, while others are dedicated statements, like the if-then-else and case ones; the last one is very important because it can specify the states of Finite State Machines (FSM). A FSM is the central module of the test equipment development, so a general description of it is now given.

A general model for Finite State Machines is reported in the left of Fig. 3.2: it contains two banks of flip-flops for state and output memorization and two asynchronous logic blocks to state and output computation. State and output processes only consist of flip-flops, delta process determines next state from the inputs and present state, while lambda determines the outputs. Following the general model, usually in VHDL each block is represented by a process or, in some cases, by concurrent statements; there is also the possibility of using a single process including all the statements in it, or some intermediate solutions. delta and lambda processes have inputs and previous state in the sensitivity list, because they have to determine their outputs when one of their inputs change; generically, they consist of a case statement, which selects different behaviours depending on inputs and previous state. Starting from the general model, two formal models exist, Mealy and Moore [35]; in the first one, outputs are function of both inputs and previous states, while in the second are functions of only state. In Mealy model, then, delta and lambda processes are unified in a single delta lambda process, as it can be seen in the right of Fig. 3.2.
Chapter 4. Test Set-Up

TELDES and InterTEL boards required two test set-ups, that are shown in Fig. 4.1 and Fig. 4.3; the elements included in the two set-ups are similar and are useful in simulating other boards behaviour and providing information needed for the tests. Beside a TEL62 simulator for TELDESs and InterTEls, some ancillary electronic boards provide signals and clocking and routing features to the boards.

The test equipment has been designed in accordance with a list of tests specifically developed for the two boards, in order to create a procedure that has to be thoroughly followed. The aim of the tests is to check signals integrity, clock parameters, Packet Error Rate (PER) and Bit Error Rate (BER) of the boards.

The DE2-115 is the main device of both the set-ups: it generates the data that have to be transmitted in order to fulfil the whole list of tests, and receives the data stream from the TEL62 simulators in order to carry out the Packet Error Rate test. A reliable firmware is then mandatory in order to guarantee the success of the test procedures.

4.1 TELDES and InterTEL Test Set-ups

The tests of TELDESs and InterTEls required two TEL62 simulators, due to the unavailability of the original boards and to the complexity that such tests would have had with them. Therefore, the Electronic Service of INFN Perugia developed such boards, including the main functions of a TEL62, together with additional features specifically needed for the tests. Fig. 4.1 shows the test set-up for TELDES, together with its block diagram, where the test board is referred to as motherboard. It has two QTS connectors where TELDESs are plugged in order to receive power supply, both 3.3 V and 5 V voltages, and external clock. The DS92LV16 recovered clock and deserialized data also pass through the QTS and are sent to sixteen 40-pin connectors, where a flat cable with forty wires links them with the FPGA 40-pin expansion connector.
The CREAM simulator consists of five boards:

- Two DS92LV18 Evaluation Kit [36]: thanks to the same packaging of the two devices, this boards have been adapted for DS92LV16 SERDES chips. The Fig. 4.2 shows the configurations of the four pins that DS92LV16 does not use as signal pins and basically two of them are connected to power supply and ground, while the other two are rise/fall clock edge control pins. A 40-pin connector receives 16-bit LVTTL/LVCMOS words from FPGA and sends them to the DS92LV16, that deserializes data and frames it with the embedded clock; the LVDS signal is routed out the board via two SMA connectors. All the lines are controlled in impedance, usually at 50 Ω, but it is worth noting that the input lines are controlled at 60 Ω: this caused some issues in the communication between the board and the FPGA that will be taken in account in the next chapters.
Another issue that has to be taken into account is the synchronization one, i.e. the rising edge of each input clock period must be in the time range where input data are stable, in order to have the correct sample position.

- Two DS15BA101/DS15EA101 Evaluation Kits [38]: each board features a DS15BA101 and a DS15EA101, and each device receives and sends signals via two SMA connectors. In the set-up, the DS15BA101 receives the LVDS data stream by the DS92LV16 and drives the correspondent output; the output voltage can be customized via on-board jumpers, which has been settled to 400 mV.

- One SMA-to-RJ45 adapter: it is an adapter that receives the two LVDS stream via 4 SMA connectors and sends them to a RJ-45 connector, in order to use two of the twisted pairs of the cable connected to it.

Because only a 40-bit connector is included in the DE2-115, a patch board is used for routing signals to and from FPGA: it consists of three 40-bit connectors, where the FPGA, CREAM simulator and TELDES motherboard are connected. The pins of the FPGA connectors have been divided in left and right sets, using the first as output pins and the second as input ones; the two sets, then, have been connected respectively to one of the other 40-pin connectors of the patch board. The unused pins have been connected to ground in order to improve signals integrity and to lower cross-talk issues in the flat cables. A Lecroy ArbStudio waveform generator is the last component of the set-up, used as a clock generator; in particular, the first output has been connected to the FPGA and to one of the DS92LV18 evaluation kits, while the second to the other evaluation kit and to an
input of a *Teledyne Lecroy* WavePro 735Zi oscilloscope, in order to have the proper trigger signal.

The FPGA sends Pseudo Random Bit Sequence (PRBS) data to the CREAM simulator via the patch board, that generates the LVDS data-stream and drives a Cat. 6 cable; the other end of the cable is connected to one of the 8 channels of a TELDES, that equalizes the signal and deserializes it to a 16-bit word that is sent to the motherboard. Via the patch board, then, data come back to the FPGA, where they are used for further analyses.

For what concerns the InterTEL test set-up, it is composed only of the motherboard and the FPGA, as it can be seen in Fig. 4.3.

![Fig. 4.3 - Picture and block diagram of the InterTEL test set-up.](image)
Because InterTEL boards includes two oscillators with two clock distributors, the 60-pin connector sends a clock output to the TEL62, that is used to drive the DE2-115 during the tests; due to this, the clock generator is no longer useful for the set-up. Furthermore, because InterTEL embeds both the serializer and deserializer, data are sent by the first one and received by the second, eliminating the need for the patch board and the SERDES and DS15BA101/DS15EA101 evaluation kits as well. The InterTEL motherboard has the same features as the TELDES one, plus others specifically included for InterTEls: there are three switches for REN, DEN and SYNC controls of the SERDESs, a 14-bit connector and a system that allows a static test for the boards. Basically, a static test consists in sending the same 16-bit word over time and checking if it is correctly received by the same board; four digit-selectors are used for generating the 16-bit that has to be transmitted, while thirty-five dipswitches route the signals from the FPGA or from the digit-selectors. Four seven-segment displays show the received word, in order to verify if it is correct or not. The 14-pin connector, instead, is used to read and write control signals to and from the FPGA, while data and recovered clock pass through a 40-pin connector, as in the TELDES case. A clock signal is sent to the DE2-115, where PRBS data are generated and sent to the serializer in the InterTEL through the motherboard; the LVDS stream passes through a DS15BA101, a Cat. 6 cable and a DS15EA101 before reaching the input of the deserializer, where it is deserialized and sent back to the DE2-115 in order to carry out some tests.

4.2 Tests List and Description

A list of tests was produced together with the INFN Perugia section and is reported in Table 4.1. Because it defines the names of the tests, the number of the boards that have to be tested and the specifications ranges in which the results have to belong to, this list has been followed as a guideline for the overall assessments of the two boards. In addition to these tests, the manufacturer of TELDESs and InterTEls did a substrate check before their assembly and final product validation by visual inspection before their delivery, in order to guarantee the validity of the PCB production. The first test is the Visual Inspection, that is an accurate check of the devices on the board, and in the TELDES case it is also checked if internal clock circuitry is enabled. The boards work with two power supplies, that are one for the PLLs of the SERDESs (VDD2) and one for the remaining devices of the board (VDD).
Table 4.1 - Tests list relative to TELDES and InterTEL boards; TCLK jitter refers only to InterTELs, while External Clock Test to TELDESs.

<table>
<thead>
<tr>
<th>Test type</th>
<th>Batch acceptance</th>
<th>Sample acceptance</th>
<th>Operational specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Inspection</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>VDD: Supply Voltage [V]</td>
<td>*</td>
<td>3.15</td>
<td>3.45</td>
</tr>
<tr>
<td>VDD2: PLL Supply Voltage [V]</td>
<td>*</td>
<td>3.15</td>
<td>3.45</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\frac{T_{\text{REFCLK}}}{T_{\text{TCLK}}}$: Ratio between the periods of reference clock (REFCLK) and transmission clock (TCLK)</td>
<td>*</td>
<td>0.95</td>
<td>1.05</td>
</tr>
<tr>
<td>DC_{\text{REFCLK}} : REFCLK Duty Cycle</td>
<td>*</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>TT_{\text{REFCLK}}: REFCLK Transition Time between 10% and 90% [ns]</td>
<td>*</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>TCLK Jitter [ps]</td>
<td>*</td>
<td>-</td>
<td>80</td>
</tr>
<tr>
<td>Static Test</td>
<td>*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PERT: Packet Error Rate Test</td>
<td>*</td>
<td>-</td>
<td>$1.12 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>Eye Diagram Test</td>
<td>*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>External Clock Test</td>
<td>*</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Both supplies are then measured, together with the power consumption of each board, while the voltage range derives from the SERDES datasheet [12]. The value of the VDD is a critical point of the assessment: indeed, in the experimental area, due to a large number of devices, the supply voltages only reached 3.14 V, that is lower than the value specified in the table. Increasing the main power supply to 3.45 V, however, the voltage to TELDESs can reach 3.22 V, but in order to consider a worst case scenario the tests have been executed with 3.15 V measured at TELDES. The clock parameters range also derives from DS92LV16 datasheet [12], where the ratio between REFCLK and TCLK, the duty cycle and the transition time of the REFCLK and the maximum jitter allowed for the TCLK are specified. The measurement of these parameters have been carried out with the WavePro 735Zi oscilloscope. The Static Test consists in sending the 16-bit word corresponding to the hexadecimal x"FFFF", i.e. a word with all ones, in order to search for main bugs or connectivity issues. This test is carried out differently for TELDES and InterTEL: in the first case, the x"FFFF" word is sent by the FPGA, while in the second by
the dedicated circuitry of the motherboard. The Packet Error Rate Test is a measure of the rate of wrong 16-bit words in a communication session; the upper bound has been calculated taking into account the maximum rate of wrong L0 trigger of the whole experiment, that is $10^{-2}$. Considering all the detectors, except the LKr electromagnetic calorimeter, without errors, and that the trigger is invalidated even though only one of the 896 output words of the CREAMs is wrong, the upper limit results to be $1.12 \times 10^{-5}$ under the hypothesis of statistically independent channels. From the cable extender chipset and SERDES datasheets, however, the expected PER is far below the needed threshold, that is in the order of $10^{-11}$; the PERT, then, would require a large amount of time per channel, becoming a critical parameter. Because an accurate estimation is not required, however, only an upper limit for the PER has been estimated, in order to reduce the time of the test. If the PER is constant during the whole test and the 16-bit words are considered as independent, the number of errors results to be a binomial random variable, which can be approximated as a Poisson's one in the case of a few errors in a large number of transmitted words. The probability of having zero errors on $n$ received words is given by Eq. 4.1, while the probability that PER is lower than a fixed threshold ($\gamma$) given $n$, is called Confidence Level (CL) and can be computed by Eq. 4.2.

$$\text{Prob}(\varepsilon = 0) = p_n(0) = \binom{n}{0} p^0 (1-p)^{n-0} = (1-p)^n \quad 4.1$$

$$\text{CL} = 1 - \text{Prob}(\varepsilon = 0) = 1 - (1-p)^n \quad 4.2$$

From the equations above, $n$ can be computed for a wanted CL=0.99 and an upper limit $\gamma=3 \times 10^{-9}$; the threshold is smaller than the requested one because the corresponding test time per channel is around forty seconds, which is still a reasonable time. Longer term tests, lasting eight hours, have also been carried out on eleven InterTELs, that is 50% of the production, in order to have a lower $\gamma$, reaching $\gamma=4.19 \times 10^{-12}$. For what concerns TELDESSs, then, a twenty hours per channel test has been carried out on a board, in order to check its behaviour on longer periods and in real operational conditions. The Eye Diagram Test is a pass-fail test used in serial data communication assessment, in order to check the noise level and jitter of the signal. An acceptance mask is specified using the requirements of the devices that can be found in their datasheet, which in our case is the DS92LV16 datasheet: the height of the mask is chosen as the minimum signal level that the LVDS input can accept, while its width comes from the maximum acceptable jitter of the same port. The signal is valid only if it has no intersection with the acceptance mask, as can be seen in Fig. 4.4; the test has been carried out using the Serial Data Analysis (SDA)
III tool of the 735Zi oscilloscope on seven TELDESs and eleven InterTELs, which are, respectively, 10% and 50% of the overall production. Finally, TELDES boards will be used with an external clock coming from the TEL62 during their normal activity; because of this, the External Clock Test has been executed on ten boards, that is 14% of the final production, in order to check if their behaviour is correct.

Fig. 4.4 - Eye Diagram Test showing the acceptance mask.
Chapter 5. FPGA Test Equipment

The main topic of this thesis work was the development on FPGA of the test equipment both for TELDES and InterTEL boards, in order to fulfil the procedure described in the previous chapter. The design was carried out following the flow described in Fig. 3.1, with the difference that each block was simulated only with Behavioural simulation, while time issues were detected via the Quartus II TimeQuest Timing Analyzer tool. Once loaded in the FPGA, the overall system was checked with SignalTap II Logic Analyzer tool, in order to verify if it had the same behaviour as in simulations. The firmware was designed and optimized both for the Cyclone IV and Stratix III FPGAs: besides the tests carried out through the DE2-115 board, the test equipment was loaded in a TEL62 SL FPGA, in order to verify the InterTEL behaviour in its actual work environment. This integration work was carried out in the experimental area at CERN.

The chapter is organized in a hierarchical structure, following the VHDL paradigm: the description starts with the maximum abstraction level, i.e. the Top Module, in order to comprehend the overall functionality of the system, and continues with the specific implementation of the blocks. All the code is reported in Appendix A.

5.1 Top Module

Fig. 5.1 shows the block diagram of the whole firmware, which has two main tasks: generate a data stream for the test set-ups and compare the received words with reference words, in order to estimate the Packet Error Rate. The Top Module, written in VHDL, is named teldes_prbs_gen and has two main functions too: control the DE2-115 tools and instantiate the main blocks of the firmware. Beside the FPGA, the DE2-115 devices used are only the SMA clock input, LEDs and push-buttons, in order to have visual information on the state of the firmware and to control its reset conditions. In particular, Key 3 is used for resetting the PLL, while Key 0 for resetting the rest of the modules. Six of the nine
green LEDs show the state in which the internal FSM is, in order to have better control on the test process and to face possible problems that can show up. The ninth green LED, instead, is used as an indicator for the PLL loss of lock over the clock input. The main function of red LEDs, instead, is to show if there are issues on the internal devices, showing if overflows of memories and counters occur and if there are errors on consecutive received words. The other fifteen LEDs indicates whether the PER test is finished or not.

The PLL is intended as a clock buffer, so it takes the 40-MHz clock coming from the SMA connector as input and has a clock output, named \textit{clk\_data}, with the same frequency and phase of the input one. As already said, its reset port is connected to the push-button 3 and it drives a green LED in order to show whether it is locked or not to the input clock.

The other two modules are a ROM memory, named \textit{ROM\_circ}, and the module for the PER test, named \textit{comparator\_top}. The first is a 2-port ROM that generates two independent data streams, one for the output port of the FPGA and one for the \textit{comparator\_top}; its task is to generate the words that have to be transmitted to the motherboards and to supply the reference words for the PER test. It has been implemented using an \textit{Altera} MegaFunction for the ROM itself, in addition to some ancillary circuitry implemented to adapt the length of the words of the MegaFunction to the ones needed for the rest of the firmware. A detailed description of the module is given in the paragraph 5.2.
It takes in input two 17-bit addresses, one for each port. For the port relative to the FPGA output, a counter synchronous to \textit{clk\_data} has been implemented in \textit{teldes\_prbs\_gen}, which counts one at every rising edge of the clock. It only stops by pressing the reset button, when it is also restored to the initial value, i.e. 0. Because of this, the FPGA continuously sends data either to the CREAM simulator, and therefore to TELDESSs, or to the InterTEls motherboard. The other address port is driven by \textit{comparator\_top}, thoroughly described in the paragraph 5.3.

In a general description, however, \textit{comparator\_top} is the module responsible for the PER test: it compares the received words and the correct ones, in order to count the number of wrong bits in each word and the number of wrong words in the communication session. Its results are shown in two signals, \textit{errors} and \textit{per}, that are left unconnected; in order to prevent the compiler from synthesizing away these signals, the \textit{keep} VHDL synthesis attribute has been used. It can only be used on signals, in order to direct the Analysis\&Synthesis process to keep them intact. It must be declared and stated as true, as shown in the code below:

\begin{verbatim}
attribute keep : boolean;
attribute keep of errors : signal is true;
attribute keep of per : signal is true;
\end{verbatim}

With this attribute, \textit{errors} and \textit{per} are not synthesized away and can be monitored using the SignalTap II Logic Analyzer.

The assignments included in the .qsf file refers to \textit{teldes\_prbs\_gen}: indirectly, then, all the properties of the firmware ports must be specified inside the highest abstraction level, i.e. the Top Module. The first property specified for each port is the pin to which it is connected. Because the Cyclone IV FPGA pins are connected to specific DE2-115 devices, ports have been connected to the external pin following the DE2-115 User's Manual [22] and using the Pin Planner tool. This is a graphical tool used for pin property assignments, as an alternative of the Assignment Editor and the .qsf file; it provides an overall view of the properties, as can be seen in Fig. 5.2. Pin assignment can be executed in two ways, that are the direct assignment of the pin to the port or the assignment to a general I/O bank, leaving the choice of the pin to the compiler. In the specific, the method used was the first one, in order to have more control over the process. Other properties that are specified in the .qsf file are the I/O standard, the current strength and the slew rate of the pins. The I/O standard specifies the reference voltage and the standard levels that a pin has to satisfy; the current strength indicates the maximum current supplied by the pin, while the slew rate is
controlled by an index indicating the speed of the transitions: higher index means a faster signal transition. The Tcl commands for the properties up to now defined are reported below, where pin, port and standard have to be substituted by the actual name of the corresponding pin, port and standard:

```
set_location_assignment pin -to port
set_instance_assignment -name IO_STANDARD "standard" -to port
set_instance_assignment -name CURRENT_STRENGTH_NEW value -to port
set_instance_assignment -name SLEW_RATE index -to port
```

An important constraint that has to be respected is that pins belonging to the same I/O bank must have the same reference voltage, that can be in the 1.2-3.3 V range for the specific FPGA.

![Fig. 5.2 - Quartus II Pin Planner tool.](image)

Two assignments used for the project are the fast_input_register and fast_output_register ones, used to activate the registers in the I/O blocks. When included, they implement a register that has a fast direct connection to the pin, improving I/O timing performance and signals integrity. They have to respect the following syntax:

```
set_instance_assignment -name FAST_INPUT_REGISTER ON -to port
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to port
```
and they must be related to signals that feed or are fed by a register, otherwise they are ignored. In the test equipment project, the ports affected by these commands are the input and output data, that are the two 16-bit standard logic vectors named, respectively, \texttt{teldes_GPIO_in} and \texttt{teldes_GPIO_out}. The former is registered synchronously to the DS92LV16 deserializer recovered clock, while the latter to \texttt{clk_data}. The need for the two assignments comes from two issues that were taken into account, that are synchronization and signals integrity. The output data that have to be read by the serializer must be stable when a rising edge of the latching clock occurs: this requires a precise synchronization between clock and data. The compiler, however, optimizes the system with many algorithms, and at least one of them takes into account the placement of the devices, in order to minimize latencies and thus speed up the system. Because of this, the delay from the external pins and the internal circuitry can be different between different compilation processes and between pins referring to the same \texttt{std_logic_vector}. The output registers are then useful because they equalize the delay between different pins and different compilations, in order to have a stable delay over position and time. The synchronization process itself has been executed in collaboration with the INFN Roma Tor Vergata group and changes depending on whether TELDESs or InterTELs are taken into account. For TELDESs, the synchronization is achieved exploiting the two independent channels of the clock generator and the stability of the signals delay. As a reminder, one of the clock channels was connected to the FPGA and to one of the DS92LV16 evaluation kits, while the other to the second evaluation kit and to the oscilloscope. If the evaluation kit that has the same clock of the FPGA is only used as interference to the main communication link in order to take into account the possible effects of the cross-talk, the clock phase of the other channel can be independently modified in order to achieve the synchronization, without affecting other aspects of the set-up. Ideally, the optimum phase would be 180 degrees, but data and clock propagate over cables of different lengths. In order to take also the cables length into account, the phase can be found by observing one of the bits of the output data and the clock at the evaluation kit input with two oscilloscope channels, by gradually increasing it until the rising edge of the clock is in the best position. In a 40-MHz clock domain, the best position means when the rising edge of the clock occurs 12.5 ns after and before two data transitions. Fig. 5.3 shows the correct synchronization between \texttt{teldes_GPIO_out} and clock measured at the input of the DS92LV16, with a phase difference of 120 degrees.
This method could not be applied to InterTELs synchronization, because the clock phases are established by the oscillators included in the boards. Nevertheless, the FPGA and the deserializer are connected to the same clock, so a phase difference can be introduced exploiting the PLL module: using the same empirical method, the phase of \( clk_{data} \), which drives the output registers, was modified until the optimum phase was found. Fig. 5.4 shows the results of the InterTELs synchronization using a phase difference of 90 degrees. For what concerns the input synchronization, the deserializer sends the data together with the latching clock, i.e. the recovered clock from the data stream, already with the correct phase relation. The implementation of the input registers, however, is mandatory due to the different possible paths coming from the optimization algorithms, that could generate different delays.
The other issue taken into account was the integrity of the signals, that have to pass through the ribbon connector, which has an impedance of 110 $\Omega$, while the FPGA, the DS92LV16 evaluation kits and the motherboards are designed with 50 $\Omega$ impedance-controlled lines. Signals degeneration is caused by their reflection due to this mismatching, together with noise and cross-talk interference. In addition to this, none of the forty pins of the FPGA connector can be connected to an internal PLL: the recovered clock, then, can not be regenerated by it and must be used as it is. This integrity issue is not completely solvable, but it affects only the test set-up, while it does not affect the boards on their work environment. Therefore, two solutions have been adopted to reduce it to a non-essential level: first, the unconnected wires of the ribbon cable were connected to ground in order to reduce the interferences between adjacent wires, and second the input and output registers had been implemented. For the input data, the registers allows to store the 16-bit words as close as possible to their source, where they are not too much affected by the noise, while the output data have the maximum signal strength when they are injected to the ribbon cable. Fig. 5.5 compares signals integrity before and after the solutions adopted.

![Fig. 5.5 - Comparison between the same signal before and after the solutions adopted to guarantee their integrity.](image)

An instance of the In-System Sources and Probes editor was implemented for remotely resetting the FPGA, in order to allow the PER test to be carried out from Perugia when the test set-ups was in Geneva. The reset signals are then two logic ors between the push-buttons and the signals controlled by the In-System Sources and Probes editor. In the
InterTEL firmware, an additional instance was implemented, in order to control the DEN and SYNC ports of the SERDES pair.

Timing constraints of the project also refer to the Top Module, and are included in the .sdc file. Although many constraints can be introduced in order to give information to the compiler on every specific signal, only clock constraints have been used. This allows a check of the firmware, in order to guarantee its behaviour for the requested clock frequency, while the external timing synchronization is left to the other methods illustrated above. Also .sdc files respect the *Tcl* syntax, and it is possible to create external clocks constraints or automatically derive the constraints for the PLLs clock outputs with the following commands:

```plaintext
create_clock -name "constraint" -period period [get_ports clk_port] 
derive_pll_clocks
```

5.2 ROM_circ

This component generates the two data streams of output words and reference words, and its architecture is shown in Fig. 5.6.

![Fig. 5.6 - ROM_circ architecture with the ROM: 2-port with two independent output and addresses ports.](image)

It is composed of three blocks, that are a dual-port ROM memory and two multiplexers, one for the output stream and one for the reference words. The ROM memory is an instance of the *ROM: 2-port* [39] MegaFunction, which has two independent address ports for read operations, easily called A or B port. These address ports are registered, while the output ports can be registered or not. It can manage one or two clock domains, whether the ports with different clocks are the A and B blocks or the address and output ones, although this feature is not used in the test equipment. It accepts ports width in the 1 to 288 range, while the maximum number of words that can be contained is 65536. This is an important constraint, because the ROM has to contain the PRBS sequence, which is composed of 131071 16-bit words. A configuration of 65536 32-bit words has
therefore been chosen, in which a single 32-bit word contains two 16-bit word of the PRBS. *ROM_circ* have two 17-bit address ports, assigned in the following way: the first sixteen most significant bits are connected to the ROM address ports, while the least significant bit to the selection inputs of the multiplexers. When read from the memory, each word is temporarily stored in a 32-bit register and the multiplexers decide which words have to be sent to the outputs.

![Image](image_url)

**Fig. 5.7 - ROM_circ** behavioural simulation carried out with ModelSim software. The names of the signals correspond to the name of Fig. 5.6.

In the simulation in Fig. 5.7 it can be seen that *ROM_circ* will provide the exact requested word by simply incrementing the external address ports by one each time needed, both for A or B output. The *ROM: 2-port* is pre-loaded with the words included in a Memory Initialization File (.mif), which is an ASCII file that specifies the initial value for each address. After a preamble in which the width and depth of the memory and the radix of the addresses and data are specified, it includes the data with some possible syntaxes. In the first syntax, each address is linked to its content by colon, as in the example below

```
01 : 01101001;
```

Another syntax allows the definition of the same value for multiple addresses

```
[0..7] : 10010110;
```

which assigns the value "10010110" to the addresses that go from 0 to 7. This last syntax was used for the TELDESs static tests, in which all the words have to be x"FFFF", while the first one for the PRBS tests. A .mif file can be also loaded in a memory block through the In-system Memory Content Editor, allowing the change of the memory content while the system is already loaded in the FPGA.

### 5.3 COMPARATOR _TOP_

As can be seen in Fig. 5.1, this module only interconnects its sub-blocks, which are a
dual clock domains FIFO memory used as a buffer, a module containing the comparator of the received words and the reference words in order to count the number of errors and a FSM that has control functions.

### 5.3.1 rdata_buf

This is the dual-clock FIFO memory provided as a MegaFunction [40] from the Quartus II software. It is a parameterizable function used as a buffer for the data received from the TELDESs and InterTELs motherboards, before their comparison in the comparator module. The MegaFunction allows three specific functions, that are:

- **SCFIFO**: it is the simplest function, with the input and output ports that are of the same width and synchronous to the same clock. SCFIFO is the common implementation of a First In First Out memory.

- **DCFIFO**: input and output ports are synchronous to two clock domains, although they are of the same width. In addition to the FIFO memory, then, circuitry that prevents metastability issues is included, allowing the function to be used as a buffer and as a clock-domain interface. This is the configuration used in the test equipment, and a ports outline is shown in Fig. 5.8.

- **DCFIFO_MIXED_WIDTH**: it is a dual clock function with ports that have two different widths.

![DCFIFO ports outline](image)

*Fig. 5.8 - DCFIFO ports outline.*

*rdata_buf* is a DCFIFO with 32768 16-bit words, with the input synchronous to the recovered clock and the output synchronous to *clk_data*. Such DCFIFO is then used to separate the two clock domains in order to make them independent from each other: data
coming from TELDES or InterTEL motherboard are written synchronously to the recovered clock, while they are read by the comparator synchronously with the \textit{clk\_data}, avoiding errors and metastability issues. The alternative to the DCFIFO was to use the recovered clock coming from the DS92LV16 as the clock for all the modules of the FPGA. Because this clock can not be connected to a PLL input and because it is not as stable as the external one, this implementation would have been less efficient and reliable than the one with the DCFIFO. The two clocks are connected to \textit{wrclk} and \textit{rdclk}, where the \textit{wr} and \textit{rd} prefixes represent the signals that are synchronous to writing or reading clock. \textit{data} and \textit{q} ports were settled as 16-bit ports in order to be connected respectively to the deserializer output and to the comparator input. Full and empty signals are available both for writing and reading domains, each one with the respective prefix, but only \textit{rdempty} and \textit{rdfull} were used. The \textit{wrreq} port has to be driven high when a valid word is available at the \textit{data} port; because the equipment does not know when valid words are available, this port was connected to a negative version of the reset signal. This means that the FIFO writes each word since the reset signal went low, even if it is an invalid word; it is up to the \textit{FSM\_control} to identify and discard the invalid words before they are counted as wrong words. For what concerns the \textit{rdreq} port, it has two possible behaviours, depending on whether the FIFO is in Normal or Show-Ahead Synchronous FIFO mode. In the first case, the data becomes available at the \textit{q} port after \textit{rdreq} is asserted, and the \textit{rdreq} actually represents a read request. In the second case, instead, the FIFO automatically outputs the first valid word and when \textit{rdreq} is asserted it outputs the next valid word; in the Show-Ahead mode, then, \textit{rdreq} acts as a read acknowledgment. \textit{wrusedw} and \textit{rdusedw} indicate the number of used words in the memory, but are not exploited in the test equipment. All the reading clock domain signals are controlled by the finite state machine in accordance with the specification included in the datasheet [40].

5.3.2 comparator

Fig. 5.9 shows the schematic block of the \textit{comparator} module, that can be divided in seven main blocks, all synchronous to \textit{clk\_data}:

- 16 xors that form the comparator itself, comparing bit-to-bit the received and the reference words. Each xor will return '0' if its inputs are identical and '1' if not, and their output can then be used to count the number of errors.
- 16 40-bit counters [41], each one counting the number of errors in a specific bit position, also named positional counters. Each count enable port is connected to
Chapter 5. FPGA Test Equipment

the logical and of the respective xor and a control signal driven by the finite state machine, named \textit{count\_en}. When it has a high value, an error occurring in a certain position will drive high the and output and consequently the count enable port. Synchronously to \textit{clk\_data}, then, a counter will count an error in the clock cycles where both the relative xor output and the \textit{count\_en} port are high.

- A 40-bit counter counting the total number of words compared in a valid communication session. Its count enable port is connected to the signal \textit{count\_words}, controlled by \textit{FSM\_control} and asserted when valid words are identified.
- A 40-bit counter to count how many wrong words are received. It is an instantiation of the same counter of the positional ones, synchronous to \textit{clk\_data} too and with the count enable port connected to a logical or of the xors outputs. It will count one if at least one of the bits of the received word is wrong and the \textit{count\_word} signal is asserted, giving a measure of the PER in a valid communication session. The or of all the sixteen words, then, is connected to the \textit{comp\_PER} output which is used by the control module.
- A 18-bit logical or taking in input the carry out port of all the counters; its output indicates if an overflow of the counters occurred during the session.
- 16 40-bit load registers [42] with buffer tasks. When the communication session ends, data from each positional counter output are stored in the relative register, which maintains the value until the beginning of a new test.
- A parallel adder which sums the output of the positional counters, giving the total number of bit errors occurred in the communication session. It has been implemented with the \textit{parallel\_add Altera} MegaFunction [41], with sixteen 40-bit input and, therefore, a 44-bit output. It takes thirty-two \textit{clk\_data} clock cycles to produce its output, while its inputs are held by the load registers.

The number of total errors are used for the BER estimation, while the number of wrong words for the PER one; the number of compared words in the overall session is used for the same purposes. Together with this information, control signals are generated by the module too, used for the synchronization of the two input data streams and to indicate if an overflow occurs.
Fig. 5.9 - Simplified schematic block of the comparator module.
5.3.3 FSM_control

*FSM_control* is a finite state machine that controls and synchronizes the other blocks of the test equipment, and it is also referred to as control module. It is a Mealy-model FSM implemented with only two processes. As can be seen in Fig. 3.2, the Mealy model foresees a unified *delta lambda* process both for states and outputs computation, and two processes, *state* and *output*, for storing the present state and the present output. These processes have been unified in a single process, named *sync_proc*, that consists only of a D flip-flop for each of the output and of the state signal. This process synchronizes the asynchronous signals coming from the *delta lambda* process and also manages the reset of the output ports, when the reset button is pushed. Because it is a synchronous process, only the clock signal appears in its sensitivity list. The *delta lambda* process, named *state_decode_proc*, is instead an asynchronous process that generates the combinatorial functions to compute next state and outputs from the inputs and the previous state. Its sensitivity list includes all the signals that have to be checked in order to allow a state transition when at least one of these signals change its value.

![FSM_Control State Diagram](image)

*Fig. 5.10 - FSM_control state diagram.*
A type for the names of the states has been declared, in order to improve the code readability: the FSM states can be indeed recalled with their proper name, instead of an integer number. Such procedure is also recommended by Altera [43] in order to allow Quartus II to better recognize and optimize the FSM. Besides the clock and reset signals, the inputs are the FIFO full and empty flags, the total number of compared words and \textit{comp\_PER}, which is the signal that indicates whether the received word contains errors or not. The output ports include the control signals of the FIFO, the \textit{comparator} and the ROM modules and a flag indicating the current state of the machine. It controls the clear and read-enable signals of the FIFO, the address port of the ROM, the clear and enable signals of the counters, registers and adder of the comparator and, finally, the load signal of the registers.

Fig. 5.10 shows the FSM state diagram, for a total count of seven states; these are extended states, meaning that the FSM can remain in the same state while changing its output. As an example, an extended state could be a state containing a counter, with different outputs for each value of it. Ideally, each of the configurations of the signals has to form a specific state, but they have grouped together in order to simplify the machine.

The list below illustrates the behaviour of the FSM in each state:

- The first state is \textbf{RESET}, which resets all the internal signals to a known state; the internal signals are the signals upstream the flip-flops of the \textit{sync\_proc}, while the outputs are named external signals. All the control signals are settled to the value 0, except for the clear signals, that are driven high. This state can be entered only via the reset button and lasts only one clock cycle.

- The FSM goes in the \textbf{IDLE} state once the \textbf{RESET} state is ended, i.e. all the modules of the firmware are in a known state, and remains in it until a word is written in the FIFO memory. This change is checked with the empty signal of the memory, which is high when the memory is empty and low when it is full. Such port, named \textit{rdata\_rdempty}, is driven low with a precise latency, defined in the FIFO datasheet; for a show-ahead dual-clock FIFO, the latency is two writing clock cycles plus a reading clock cycle for each synchronization stage of the FIFO. The synchronization stages are the steps through which the input data have to pass through in order to avoid metastability issues. In the \textit{rdata\_buf} case, therefore, the total latency is two writing and three reading clock cycles. When \textit{rdata\_rdempty} goes low, then, the FSM turns into the \textbf{SYNCHRONIZATION} state, otherwise it remains in the \textbf{IDLE} one.
In the SYNCHRONIZATION state the FSM has to synchronize the FIFO and the ROM data streams because they reach the comparator inputs following two different paths and having different latencies. The signals rdata_rdempty and rdata_rdfull are both checked in order to verify if the FIFO goes in empty or full state: if an empty occurs, the FSM goes in the IDLE state, otherwise it goes in the FULL state.

| Name          | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| LED(3)[2]     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(3)[1]     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(3)[0]     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(2)        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(1)        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(0)        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(0)[8]     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LED(0)[7]     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Fig. 5.11 shows the behaviour of the signals of interest in the SYNCHRONIZATION; the state is extended by the cnt_sync variable, which can assume values from 0 to 15. At the beginning of the state, the reference word address is fixed to a defined value, i.e. fifty or x"32", while the rdata_rdreq signal of the FIFO is maintained high. The FIFO words are therefore discarded until there are no errors in the comparison with the fixed reference word, which is indicated by the lowering of the comp_PER signal, as showed in the clock cycle 55 of the figure. The FSM stops the reading from the FIFO and increments by one the ROM address, maintaining their values for the three following clock cycles, i.e. when cnt_sync value is 1, 2 and 3. Because the ROM has a latency of one clock cycle from the change of the address to the correspondent word, the ROM output changes when cnt_sync is 2. If comp_PER is low when cnt_sync is 3, rdata_rdreq is asserted and the address is incremented of one, changing the output words of the memories when cnt_sync is 5. comp_PER is then checked and if it is low, the signals are changed again in the 62nd cycle of Fig. 5.11. Since cnt_sync value is 9, rdata_rdreq is always asserted and the address is always incremented, in order to change the words in every clock cycle; also comp_PER is always checked, in order to be confident that the two streams are synchronized, and after
six cycles, i.e. when \( \text{cnt\_sync} \) is 15, FSM goes in the COMPARISON state if no errors were detected. If \( \text{comp\_PER} \) is high in one of the previous controls, the SYNCHRONIZATION state starts from the beginning.

- **COMPARISON** is the main state, in which the system actually compares the incoming words. The \( \text{count\_en} \) and \( \text{count\_words} \) ports of the comparator are asserted, so for each clock cycle the positional counters count the number of errors in a word, the PER counter the number of wrong words and the words counter the total compared words. The address of \( \text{ROM\_circ} \) corresponding to the reference words is incremented by one for each clock cycle, until the maximum of 131071 is reached, and the address starts back from zero. The \( \text{rdreq} \) port of the FIFO is always asserted and for each clock cycle the received word is compared to the corresponding reference word, as described in the comparator paragraph. The number of compared words is checked by the FSM, which stops the COMPARISON state once a predefined value is reached: for the forty seconds test, the threshold is \( \text{x"005F5E1000"} \), which is the 10-digit hexadecimal that corresponds to sending \( 40 \cdot 10^6 \) words per second for forty seconds. For the long term test lasting around eight hours, instead, the threshold has been fixed at \( \text{x"7FFFFFFFFFE"} \). When the words counter reaches the threshold, the counters and FIFO enable signals are de-asserted, while the counters output are loaded into the registers and the FSM goes to the SUM state. \( \text{rdata\_rdempty} \) and \( \text{rdata\_rdfull} \) are always monitored as in the previous state, in order to face the possible issues that can occur.

- In the SUM state, all the modules are disabled, except for \( \text{ROM\_circ} \) that always sends words to the test set-up and for the parallel adder. Its \( \text{clken} \) port is maintained high for thirty-two clock cycles, in order to properly sum the addendum at its input. When the sum is complete the adder is disabled and the state turns to SHOW.

- **SHOW** is a state where the FSM only maintains its outputs, in order to allow the user to read the results with SignalTap II. No transitions are foreseen from this state and the only way to exit from it is by pressing the reset button.

- **FULL** is the last state, and is accessible from IDLE, SYNCHRONIZATION and COMPARISON. In the case that FIFO assumes the full state, in order to prevent errors in the words comparison, FSM clears all the words in the memory and starts back from the IDLE state. The counters are not cleared, so that the
comparison can continue from the same point in which the full occurred; before this, the data streams are always synchronized through the SYNCHRONIZATION state.

5.4 Compilation Results

After a complete behavioural simulation of all the blocks, a first compilation process was carried out; in order to reach the requested performance, therefore, the software algorithms were tuned-up. The overall compilation process was settled to performance optimization, reserving extra effort for it and choosing speed-optimization algorithm, at the expenses of power consumption and resource optimization. In particular, for the Analysis&Synthesis process the recognition and optimization of finite state machines was activated, while the power optimization was disabled, as well as the suppression of duplicate registers. This last setting was disabled because the clear signals of the FIFO and of the comparator would be otherwise unified, even though they belong to two different clock domains, generating metastability issues. For what concerns the Fitter process, it was optimized for metastability, improving the reliability of the design by increasing its mean time before failure. One of the standard settings consisted in reducing the Fitter effort once the timing requirements were met; the other possibility, which was the preferred one, was to maintain the same effort for the whole process, although it requires a longer time.

TimeQuest Timing Analyzer takes into account the operational conditions of the board, in order to report all the probable timing issues. The conditions include the temperature of the board, the applied supply voltage and the speed grade of the silicon devices. Two worst-case conditions have been considered: the former considers slow speed grade, low voltage and 85°C temperature, while the latter slow speed grade, low voltage and 0°C. These temperatures represent the range suitable for commercial devices. For the two conditions, TimeQuest reports the maximum frequency that every clock in the design can assume, computed as the maximum frequency that every design block can manage. For the first condition, the maximum recovered-clock frequency can be 173.46 MHz, while the maximum \( \text{clk\_data} \) frequency is 970.87 MHz. In the second condition, instead, the recovered-clock can have a maximum frequency of 187.3 MHz, while \( \text{clk\_data} \) 1089.32 MHz. I/O blocks can actually manage a maximum toggle rate of 250 MHz, so the maximum \( \text{clk\_data} \) frequency is restricted to 250 MHz. Because both the recovered clock and \( \text{clk\_data} \) frequencies are fixed at 40 MHz, the FPGA design can be reliably used for the test purposes. No metastability issues was detected by the timing analyzer.
5.5 IN-SYSTEM TESTING

After its compilation, the design was loaded into the FPGA and tested in order to verify its actual behaviour. As already said, the in-system test was carried out using SignalTap II Logic Analyzer. Firstly, signals that have to be checked are added to the main window, in order to allow the tool to instantiate the requested circuitry; secondly, a sub-set of these signals is chosen as trigger signals. It is possible to specify more than a trigger condition, controlling whether each signal behaves in the requested way; when the condition is met, SignalTap II reads the data from the FPGA, sampling the signals with the clock specified in the settings.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEDG[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDG[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDE[17]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top_add/PRBS_Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...comparator/data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top_add/PER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top_add/ERRORS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>teldeg_GPIO_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>teldeg_GPIO_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>teldeg_GPIO_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>teldeg_GPIO_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDE[19]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.12 - SignalTap II acquisition showing the transition between SUM and SHOW states, here represented by LEDG[4] and LEDG[5] respectively.

Fig. 5.12 shows a SignalTap acquisition showing the transition from the SUM to the SHOW states. As it can be seen, the PER signal indicates that two wrong words were detected during the test, while the ERRORS one shows that x"C", i.e. twelve, bit errors were detected. The value of ERRORS is valid only when the parallel adder had finished the sum process, i.e. when the FSM enters the SHOW state. The software was settled to trigger when the signal corresponding to the SHOW state was asserted. This example refers to an InterTEL test, when plugged to a TEL62, as described in the paragraph below.

5.6 INTERTEL INTEGRATION

The InterTels were tested both plugged on the motherboard and on a TEL62: this allowed to verify their correct behaviour in their actual work conditions. Eleven InterTels, corresponding to 50% of the batch, were tested when plugged to a TEL62. With this test, also the ribbon cable noise and mismatching issues are removed, as a difference with the motherboard case.
Fig. 5.13 shows the TEL62 60-pin fine-pitch connector with the names of the ports of the SL FPGA to which each pin is connected. AUXBRD_CLKA is connected to the clock coming from the InterTELs clock distributor, while the AUXBRD_CLKB to the DS92LV16 recovered clock. Both of them are connected to general I/O pins of the FPGA, so they can not be inputs of a PLL. AUXBRD_CTLA and AUXBRD_CTLB are six pins dedicated to some control signals, that are the same three control signals controlled by the motherboard switches, i.e. SYNC, REN and DEN, plus three signals for monitoring the devices. Finally, AUXBRD_DATAA and AUXBRD_DATAB are thirty-two pins dedicated, respectively, to the sent and received words.

Before any test with the SL FPGA the signals integrity was checked, in order to guarantee the compatibility of the InterTELs with the TEL62; as it can be seen in Fig. 5.14, signals show a proper behaviour, with less noise and interference than in the motherboard case. For a first integration test, therefore, the same equipment as in the Cyclone IV FPGA has been loaded, in order to carry out the test procedure described above. The LVDS output of the InterTEL was connected to its LVDS input, in order to perform a closed-loop test and check each single board.
The PERT was conducted for eight hours per board, and most of them showed no errors, indicating a PER less than $\gamma=4.19 \cdot 10^{-12}$ with a CL of 99%. One of the boards showed three errors, that correspond to $\gamma=9.14 \cdot 10^{-12}$. A target PER for the InterTELs is not specified, due to the fact that the detectors have not yet decided either the topologies or the data that have to pass through the InterTELs. The PERs highlighted by the tests, however, are sufficiently low to guarantee a reliable communication between InterTELs, and with a maximum distance of around 85 meters, as stated in [18].

The future integration works, in collaboration with CEDAR, RICH and LAV working groups, foresee two TEL62s connected by two InterTELs and the use of the actual SL firmware, in order to make InterTELs ready to be used. The system that will be loaded in the SL FPGAs of the TEL62 where InterTELs are plugged will send to the InterTELs a 40-bit word clocked at 160 MHz each time a particle hits the detector, with a mean hit rate of 10 MHz. Because the clock domains of the firmware and of the InterTEL are different, two dual-clock FIFO must be implemented in order to allow the correct communication between TEL62s. The first FIFO will be similar to the FIFO implemented in the test equipment, i.e. synchronous to the recovered clock when writing and to the firmware clock when reading, while the second synchronous to the 40 MHz clock from the InterTEL clock distributor and to the 160 MHz clock, for writing and reading respectively. These two FIFOs also solve the issue of the integrity of the clocks: even though the clocks coming from the InterTELs can not be connected to a PLL, they are used only for reading from and writing to a FIFO placed near the FPGA pins. The clock signals, therefore, do not have to drive many devices or go through long distances, and are satisfactory also without the PLL. Together with the input and output FPGA registers, finally, they also guarantee the integrity of the data signals, as already described above.
Conclusion

During the thesis work, a FPGA test equipment for TELDES and InterTEL boards was designed. Its development is the result of the first project combined with the issues observed during the integration of the two boards in the experimental area at CERN, Geneva.

The tasks of the equipment were two, i.e. generating a data stream for the two boards and analyzing the received words in order to perform a Packet Error Rate Test. The data stream was generated with a 2-port ROM, that sent 16-bit words through dedicated output registers, in order to guarantee the signals integrity and the synchronization of the words with the clock that drove the DS92LV16 serializer. Such synchronization was achieved with a phase difference of 120 degrees between the clocks that fed the FPGA and the DS92LV16. Once transmitted and serialized, the words were received by TELDESs or InterTEls, that send them again to the FPGA, together with the deserializer recovered clock. Since the recovered clock pin was not connected to a PLL input and the signals integrity over the ribbon cables was not guaranteed, input registers were added to the system. Downstream the registers, a dual-clock FIFO acted both as a buffer and as a clock domains interface, in order to avoid metastability issues. The received words were compared to the reference words, also included in the 2-port ROM, in order to estimate the PER of a session of the communication. The total number of the received wrong words was stored in a 40-bit counter, while the total number of wrong bits was computed by sixteen counters and a 44-bit parallel adder. The two values were read through SignalTap II Logic Analyzer and used for the estimation of the PER and BER of the session.

Once tested, the circuitries had to be integrated in the experimental area. InterTEL boards particularly needed additional tests, in order to verify their behaviour when plugged to the TEL62s. The first integration step consisted in the electrical check of the signals, while a PER estimation was carried out by connecting the output and the input ports of
each InterTEL. The second step, that is still ongoing, consisted in connecting two or more InterTELS and verify their correct behaviour.

In closing, a reliable test equipment for the TELDES project for the NA62 experiment LKr electromagnetic calorimeter L0 trigger circuitry was designed, in collaboration with the INFN Perugia and Roma Tor Vergata groups and CERN personnel, and implemented on FPGA. The same equipment was optimized for the assessment of the InterTEL project for the CEDAR, LAV and RICH detectors, in collaboration with the INFN Firenze, Perugia and Frascati sections and University of Birmingham. The tested boards of both the projects were then successfully installed in the experimental area at CERN, Geneva. An article was submitted by the LKr working group for the proceeding of the 2014 TWEPP, and it was accepted and will be published soon. Another article was produced and it will be submitted to the IEEE NSS and MIC. Finally, two technical reports were produced and they will be published in the EDMS of CERN.
List of Figures

Fig. 1.1 - CERN accelerators complex: the path starts with linear accelerators (LINACS), continuing with Proton Synchrotron Booster (PSB), Proton Synchrotron (PS), Super Proton Synchrotron (SPS) and Large Hadron Collider (LHC), reaching energies up to 14 TeV. ................................................................. 10

Fig. 1.2 - Unification process of fundamental interactions [8]........................................ 11

Fig. 1.3 - Elementary particles and their properties (mass, charge, spin); quarks and leptons are called Fermions and are divided in three generations, formed by the particles of each column of the figure [9]................................................................. 12

Fig. 1.4 - NA62 schematic showing the position of all the detectors of the experiment, installed in the ECN3 in the North Area Facility of the SPS [10]. .................. 13

Fig. 1.5 - LKr readout and trigger systems, also showing a picture of the calorimeter before the installation ........................................................................................................ 16

Fig. 1.6 - LKr L0 processor block diagram; GbE stands for Gigabit Ethernet. ............... 17

Fig. 2.1 - Pin and block diagrams of the DS92LV16; the serializer is in the upper part of the block diagram, while the deserializer is in the lower part.................. 18

Fig. 2.2 - Datalink layer frame of the DS92LV16, with 16 data bits and embedded clock bits; start bit is stuck at 1, while stop bit at 0, forming a low-high transition used by PLLs for clock recovery......................................................... 19

Fig. 2.3 - Typical LVDS interface, with transmitter, load and balanced transmission line [15]................................................................................................................... 21

Fig. 2.4 - TI recommended settings for DS15BA101 and DS15EA101 as cable extender chipset; images of a hypothetical signal show its evolution over the system.... 22

Fig. 2.5 - Basic architecture for a FPGA, showing the elementary components............ 24

Fig. 2.6 - Two possible architectures for a simple LE, either with or without a Full-Adder. .................................................................................................................. 25
Fig. 2.7 - DE2-115 board featuring an *Altera* Cyclone IV E FPGA ......................26
Fig. 2.8 - View of the Cyclone IV EP4CE115 FPGA; blue blocks are LABs, white blocks are DSPs, green are memory blocks and brown are I/O blocks. In black, empty blocks are represented.................................................................27
Fig. 2.9 - Cyclone IV LAB architecture, including 16 LEs, local interconnects and direct link connection.................................................................28
Fig. 2.10 - Block diagram of a Cyclone IV Logic Element....................................28
Fig. 2.11 - Block diagram for a Cyclone IV I/O block; the block shown is configured as an output block, and flip-flop, output buffer and pad are highlighted in blue........29
Fig. 2.12 - Block diagram of a Stratix III Adaptive Logic Module, showing two six-input LUTs, two FAs and two registers.................................................................30
Fig. 2.13 - TEL62 [26] board used for the LKr calorimeter, with two TELDESs plugged on four QTS connectors.................................................................31
Fig. 2.14 - From the left, top and bottom view of a TELDES [26] board are shown: in the top, sixteen equalizers and deserializer can be seen, while in the bottom the Ethernet and QTS connectors.................................................................32
Fig. 2.15 - Cat. 5e insertion loss per meter as a function of the frequency [18]; 360 MHz is highlighted.................................................................33
Fig. 2.16 - Typical performance for DS92LV18 over Cat. 5e, with the cable extender chipset [18].................................................................34
Fig. 2.17 - Cat. 6 cable section.................................................................34
Fig. 2.18 - From the left, top and bottom view of an InterTEL [26] board can be seen: the top includes two DS92LV16s, a DS15EA101 and a DS15EA101, together with signal buffers and clock devices; the bottom shows the Ethernet and 60-pin fine pitch SMD connectors.................................................................35
Fig. 3.1 - FPGA design flow.................................................................38
Fig. 3.2 - General model for a FSM in the left and model for a Mealy FSM in the right [30].................................................................43
Fig. 4.1 - Picture and block diagram of the TELDES test set-up................................45
Fig. 4.2 - DS92LV18 and DS92LV16 PCB compatibility layout [37]........................46
Fig. 4.3 - Picture and block diagram of the InterTEL test set-up................................47
Fig. 4.4 - Eye Diagram Test showing the acceptance mask......................................51
Fig. 5.1 - Block diagram of the FPGA test equipment, where in orange are highlighted the modules that are clocked with the PLL outputs..................................................53
Fig. 5.2 - Quartus II Pin Planner tool. ................................................................. 55
Fig. 5.3 - TELDES synchronization with phase difference of 120 degrees; also the noise and interference on the data signal are worth noticing. ................................. 57
Fig. 5.4 - InterTEL synchronization showing the transmitting clock and the serializer input downstream the buffer; the phase difference is 90 degrees. ....................... 57
Fig. 5.5 - Comparison between the same signal before and after the solutions adopted to guarantee their integrity. ................................................................. 58
Fig. 5.6 - ROM_circ architecture with the ROM: 2-port with two independent output and addresses ports. ........................................................................... 59
Fig. 5.7 - ROM_circ behavioural simulation carried out with ModelSim software. The names of the signals correspond to the name of Fig. 5.6................................. 60
Fig. 5.8 - DCFIFO ports outline. ............................................................................. 61
Fig. 5.9 - Simplified schematic block of the comparator module......................... 64
Fig. 5.10 - FSM_control state diagram................................................................. 65
Fig. 5.11 - Detail of the SYNCHRONIZATION process showing the behaviour of the signals of interest. ................................................................. 67
Fig. 5.12 - SignalTap II acquisition showing the transition between SUM and SHOW states, here represented by LEDG[4] and LEDG[5] respectively............. 70
Fig. 5.13 - TEL62 60-pin fine-pitch SMD connector showing the connection between pins and SL FPGA ports. ................................................................. 71
Fig. 5.14 - AUXBRD_DATAB[15] signal taken upstream of the buffers of an InterTEL plugged on a TEL62.............................................................................. 72
Bibliography


[28] Telecommunications Industry Association, *Commercial Building...


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

--Top Module-----------------------------------------------
entity teldes_prbs_gen is
generic(
    word_lgt : natural := 16;
    prbs_width : natural := 17;
    prbs_lgt : natural := 131071
);
port (SMA_CLKIN : in std_logic;
    LEDG : out std_logic_vector(8 downto 0);
    LEDR : out std_logic_vector(17 downto 0);
    KEY : in std_logic_vector(3 downto 0);
    teldes_GPIO_rclk : in std_logic;
    teldes_GPIO_in : in std_logic_vector(word_lgt-1 downto 0);
    teldes_GPIO_out : out std_logic_vector(word_lgt-1 downto 0);
    teldes_GPIO_spare_out : out std_logic_vector(2 downto 0)
);
end teldes_prbs_gen;

architecture teldes_prbs_gen_behav of teldes_prbs_gen is

component pll_data port(arest : in std_logic;
inclk0 : in std_logic;
c0 : out std_logic;
locked : out std_logic);
end component;

component ROM_circ port(
    CLK : in std_logic;
    ADDRESS_TX : in std_logic_vector(prbs_width-1 downto 0);
    ADDRESS_RX : in std_logic_vector(prbs_width-1 downto 0);
    Q_TX : out std_logic_vector(word_lgt-1 downto 0);
    Q_RX : out std_logic_vector(word_lgt-1 downto 0);
);
end component;

component comparator_top port(
    CLK40 : in std_logic;
    R_CLK : in std_logic;
    RST : in std_logic;
    STATE : out std_logic_vector (6 downto 0);
);
COUNTER_OVERFLOW : out std_logic;
PRBS_ADDRESS : out std_logic_vector (16 downto 0);
PRBS_Q : in std_logic_vector (word_lgt-1 downto 0);
R_DATA : in std_logic_vector (word_lgt-1 downto 0);
ERRORS : out std_logic_vector (43 downto 0);
PER : out std_logic_vector (39 downto 0);
SEQ_ERR_FLAG : out std_logic);
end component;

component Probe_RST port (source : out std_logic_vector (1 downto 0))
end component;

signal rst, pll_rst : std_logic := '0';
signal clk_data : std_logic;
signal clk_data_locked : std_logic;
signal clk_data_locked_buf0, clk_data_locked_buf1 : std_logic := '0';
signal clk_data_lockloss : std_logic := '0';
signal address_rx : std_logic_vector (prbs_width-1 downto 0);
signal address_tx : std_logic_vector (prbs_width-1 downto 0);
signal address_tx_cnt : natural range 0 to (prbs_lgt-1) := 2;
signal q_rx, q_tx : std_logic_vector (word_lgt-1 downto 0);
signal source_sig : std_logic_vector (1 downto 0);
signal state_flag : std_logic_vector (6 downto 0);
signal CO_flag : std_logic;
signal CO_flag_buf0, CO_flag_buf1 : std_logic := '0';
signal CO_flag_led : std_logic := '0';
signal FULL_flag_buf0, FULL_flag_buf1 : std_logic := '0';
signal FULL_flag_led : std_logic := '0';
signal errors : std_logic_vector (43 downto 0);
signal per : std_logic_vector (39 downto 0);
attribute keep : boolean;
attribute keep of errors : signal is true;
attribute keep of per : signal is true;
signal teldes_GPIO_in_buf : std_logic_vector (word_lgt-1 downto 0);

begin

pll_data_inst : pll_data port map(
        areset => pll_rst,
        inclk0 => SMA_CLKIN,
        c0 => clk_data,
        locked => clk_data_locked
    );

Probe_RST_inst : Probe_RST PORT MAP (source => source_sig
    );

ROM_circ_inst : ROM_circ port map(
    CLK => clk_data,
    ADDRESS_TX => address_tx,
    ADDRESS_RX => address_rx,
Appendix A

Q_TX     => q_tx,
Q_RX     => q_rx
);

comparator_top_inst : comparator_top port map(
  CLK40    => clk_data,
  R_CLK    => teldes_GPIO_rclk,
  RST      => rst,
  STATE    => state_flag,
  COUNTER_OVERFLOW => CO_flag,
  PRBS_ADDRESS => address_rx,
  PRBS_Q    => q_rx,
  R_DATA    => teldes_GPIO_in_buf,
  ERRORS    => errors,
  PER       => per,
  SEQ_ERR_FLAG => LEDR(15)
);

pll_rst <= ((not KEY(3)) or (not source_sig(1)));
rst    <= ((not KEY(0)) or (not source_sig(0)));

address_tx <= std_logic_vector(to_unsigned(address_tx_cnt, prbs_width));

LEDG(5 downto 0) <= state_flag(5 downto 0);
LEDG(7 downto 6) <= "00";
LEDG(8)      <= clk_data_lockloss;
LEDR(14 downto 0) <= (others => state_flag(5));
LEDR(16)     <= CO_flag_led;
LEDR(17)     <= FULL_flag_led;
teldes_GPIO_spare_out <= (others => '0');

input_clock_process : process(SMA_CLKIN)
begins
  if (rising_edge(SMA_CLKIN)) then
    if (rst = '1') then
      clk_data_locked_buf0 <= '0';
      clk_data_locked_buf1 <= '0';
      clk_data_lockloss    <= '0';
    else
      clk_data_locked_buf0 <= clk_data_locked;
      clk_data_locked_buf1 <= clk_data_locked_buf0;
      if (clk_data_locked_buf1 = '0') then
        clk_data_lockloss <= '1';
      end if;
    end if;
  end if;
end process;

main : process (clk_data)
begins
  if (rising_edge(clk_data)) then
    if (rst = '1') then
      address_tx_cnt <= 2;
      CO_flag_buf0  <= '0';
      CO_flag_buf1  <= '0';
      CO_flag_led   <= '0';
    end if;
end if;

FULL_flag_buf0 <= '0';
FULL_flag_buf1 <= '0';
FULL_flag_led <= '0';
else
  teldes_GPIO_out <= q_tx;
  if(address_tx_cnt = prbs_lgt-1) then
    address_tx_cnt <= 0;
  else
    address_tx_cnt <= address_tx_cnt + 1;
  end if;
  CO_flag_buf0 <= CO_flag;
  CO_flag_buf1 <= CO_flag_buf0;
  if(CO_flag_buf1 = '1') then
    CO_flag_led <= '1';
  end if;
  FULL_flag_buf0 <= state_flag(6);
  FULL_flag_buf1 <= FULL_flag_buf0;
  if(FULL_flag_buf1 = '1') then
    FULL_flag_led <= '1';
  end if;
end if;
end process;

process(teldes_GPIO_rclk)
begin
  if(rising_edge(teldes_GPIO_rclk)) then
    teldes_GPIO_in_buf <= teldes_GPIO_in;
  end if;
end process;
end architecture;

--ROM_circ-------------------------------------------------------------

entity ROM_circ is
  port(
    CLK : in   std_logic;
    ADDRESS_TX : in  std_logic_vector(16 downto 0);
    ADDRESS_RX : in  std_logic_vector(16 downto 0);
    Q_TX : out  std_logic_vector(15 downto 0);
    Q_RX : out  std_logic_vector(15 downto 0))
  );
end ROM_circ;

architecture ROM_circ_behavioral of ROM_circ is
  component ROM_2port
    PORT(
      address_a :    IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
      address_b :    IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
      clock :        IN  STD_LOGIC;
      q_a :          OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
      q_b :          OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
  end component;
signal add_int_tx, add_int_rx : std_logic := '0';
signal mux_in_tx : std_logic_vector(31 downto 0) := (others=>'0');
signal mux_in_rx : std_logic_vector(31 downto 0) := (others=>'0');

begin
ROM_2port_inst : ROM_2port port map(
    address_a => ADDRESS_TX (16 downto 1),
    address_b => ADDRESS_RX (16 downto 1),
    clock => CLK,
    q_a => mux_in_tx,
    q_b => mux_in_rx);
Q_TX <= mux_in_tx(31 downto 16) when add_int_tx = '0' else mux_in_tx(15 downto 0);
Q_RX <= mux_in_rx(31 downto 16) when add_int_rx = '0' else mux_in_rx(15 downto 0);

latency : process(CLK)
begin
    if (rising_edge(CLK)) then
        add_int_tx <= ADDRESS_TX(0);
        add_int_rx <= ADDRESS_RX(0);
    end if;
end process;
end ROM_circ_behavioral;

--comparator_top---------------------------------------------------------------
entity comparator_top is
    port(
        CLK40 : in std_logic;
        R_CLK : in std_logic;
        RST : in std_logic;
        STATE : out std_logic_vector (6 downto 0);
        COUNTER_OVERFLOW : out std_logic;
        PRBS_ADDRESS : out std_logic_vector (16 downto 0);
        PRBS_Q : in std_logic_vector (15 downto 0);
        R_DATA : in std_logic_vector (15 downto 0);
        ERRORS : out std_logic_vector (43 downto 0);
        PER : out std_logic_vector (39 downto 0);
        SEQ_ERR_FLAG : out std_logic
    );
end comparator_top;

architecture comp_behavioral of comparator_top is

COMPONENT FSM_control
    PORT(
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        rdata_rdempty : IN STD_LOGIC;
        rdata_rdfull : IN STD_LOGIC;
        rdata_aclr : OUT STD_LOGIC;
        rdata_rdreq : OUT STD_LOGIC;
        prbs_address : OUT STD_LOGIC_VECTOR(16 DOWNTO 0);
        comp_analyzed_prbs : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
    )
end FSM_control;
Appendix A

COMPONENT comp
  GENERIC (in_width : INTEGER := 16; count_width : INTEGER := 40);
  PORT (clk : IN STD_LOGIC;
  rdata : IN STD_LOGIC_VECTOR (in_width-1 DOWNTO 0);
  prbs : IN STD_LOGIC_VECTOR (in_width-1 DOWNTO 0);
  sclr : IN STD_LOGIC;
  count_en : IN STD_LOGIC;
  count_words : IN STD_LOGIC;
  analyzed_prbs : OUT STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
  PER : OUT STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
  comp_PER : OUT STD_LOGIC;
  reg_en : IN STD_LOGIC;
  reg_load : IN STD_LOGIC;
  reg_sclr : IN STD_LOGIC;
  add_en : IN STD_LOGIC;
  add_sclr : IN STD_LOGIC;
  total_error : OUT STD_LOGIC_VECTOR (43 DOWNTO 0);
  COUNTER_OVERFLOW : OUT STD_LOGIC;
  seq_err_flag : OUT STD_LOGIC)
)
END COMPONENT;

COMPONENT teldes_fifo
  PORT (
  aclr : IN STD_LOGIC := '0';
  data : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
  rdclk : IN STD_LOGIC;
  rdreq : IN STD_LOGIC;
  wrclk : IN STD_LOGIC;
  wrreq : IN STD_LOGIC;
  q : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
  rdempty : OUT STD_LOGIC;
  rdfull : OUT STD_LOGIC)
)
END COMPONENT;

component teldes_fifo
  PORT (
  aclr : std_logic := '0';
  data : std_logic_vector (15 downto 0) := (others => '0');
  rdreq, rdempty, rdfull : std_logic := '0';
  wrreq, wrempty, wrfull : std_logic := '0';
  q : std_logic_vector (15 downto 0) := (others => '0');
  wrreq_sig : std_logic;
  ...
)
END component;

signal rdata_aclr : std_logic := '0';
signal rdata_rdreq, rdata_rdempty, rdata_rdfull : std_logic := '0';
signal rdata_wreq, rdata_wrempty, rdata_wrfull : std_logic := '0';
signal rdata_q : std_logic_vector (15 downto 0) := (others => '0');
signal wrreq_sig : std_logic;

signal comp_sclr, count_en, count_words : std_logic := '0';
signal reg_en, reg_load, reg_sclr, add_en, add_sclr : std_logic := '0';
signal analyzed_prbs : std_logic_vector (39 downto 0) := (others => '0');
signal comp_PER : std_logic := '0';
begin

controller : FSM_control PORT MAP(
  clk => CLK40,
  rst => RST,
  rdata_rempty => rdata_rempty,
  rdata_rdfull => rdata_rdfull,
  rdata_aclr => rdata_aclr,
  rdata_rdreq => rdata_rdreq,
  prbs_address => prbs_address,
  comp_analyzed_prbs => analyzed_prbs,
  comp_PER => comp_PER,
  comp_sclr => comp_sclr,
  comp_count_en => count_en,
  comp_count_words => count_words,
  reg_en => reg_en,
  reg_load => reg_load,
  reg_sclr => reg_sclr,
  add_en => add_en,
  add_sclr => add_sclr,
  state_flag => STATE
);

rdata_buf : teldes_fifo PORT MAP (
  aclr => rdata_aclr,
  data => R_DATA,
  rdclk => CLK40,
  rdreq => rdata_rdreq,
  wrclk => R_CLK,
  wrreq => wrreq_sig,
  q => rdata_q,
  rdempty => rdata_rempty,
  rdfull => rdata_rdfull
);

comparator : comp PORT MAP (
  clk => CLK40,
  rdata => rdata_q,
  prbs => PRBS_Q,
  sclr => comp_sclr,
  count_en => count_en,
  count_words => count_words,
  analyzed_prbs => analyzed_prbs,
  PER => PER,
  comp_PER => comp_PER,
  reg_en => reg_en,
  reg_load => reg_load,
  reg_sclr => reg_sclr,
  add_en => add_en,
  add_sclr => add_sclr,
  total_error => ERRORS,
  COUNTER_OVERFLOW => COUNTER_OVERFLOW,
  seq_err_flag => SEQ_ERR_FLAG
);

end comp_behavioral;

--comparator-------------------------------------------------------------

entity comp is
  generic(

Appendix A

in_width : integer := 16;
count_width : integer := 40
);
port(
clk : in std_logic;
rdata : in std_logic_vector(in_width-1 downto 0);
prbs : in std_logic_vector(in_width-1 downto 0);
scir : in std_logic;
count_en : in std_logic;
count_words : in std_logic;
analyzed_prbs : out std_logic_vector (count_width-1 downto 0);
PER : out std_logic_vector (39 downto 0);
comp_PER : out std_logic;
reg_en : in std_logic;
reg_load : in std_logic;
reg_sclr : in std_logic;
add_en : in std_logic;
add_sclr : in std_logic;
total_error : out std_logic_vector (43 downto 0);
COUNTER_OVERFLOW : out std_logic;
seq_err_flag : out std_logic
);
end comp;

architecture comp_behavioral of comp is

component counter_40bit
port(
   clock : IN STD_LOGIC ;
cnt_en : IN STD_LOGIC ;
scir : IN STD_LOGIC ;
cout : OUT STD_LOGIC ;
q : OUT STD_LOGIC_VECTOR (count_width-1 Downto 0)
);
end component;

component load_register
port(
   clock : IN STD_LOGIC ;
data : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
enable : IN STD_LOGIC ;
load : IN STD_LOGIC ;
scir : IN STD_LOGIC ;
q : OUT STD_LOGIC_VECTOR (count_width-1 Downto 0)
);
end component;

component parallel_adder
port(
   aclr : IN STD_LOGIC := '0';
clken : IN STD_LOGIC := '1';
clock : IN STD_LOGIC := '0';
data0x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data10x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data11x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data12x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data13x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data14x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data15x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data1x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
data2x : IN STD_LOGIC_VECTOR (count_width-1 Downto 0);
);
data3x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data4x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data5x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data6x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data7x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data8x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
data9x : IN STD_LOGIC_VECTOR (count_width-1 DOWNTO 0);
result : OUT STD_LOGIC_VECTOR (count_width-1+4 DOWNTO 0);
end component;

type COMP_OUT is array (15 downto 0) of STD_LOGIC VECTOR (count_width-1 downto 0);

signal error : std_logic_vector (in_width-1 downto 0);
signal en, en_s : std_logic_vector (in_width-1 downto 0);
signal count_en_vec : std_logic_vector (in_width-1 downto 0);
signal cout : std_logic_vector (17 downto 0);
signal q_count : COMP_OUT;
signal add_in : COMP_OUT;
signal comp_PER_i, comp_PER_s : std_logic;
signal count_words_s : std_logic;
signal reg_en_s, reg_load_s, reg_sclr_s : std_logic;
signal cnt_en_s : std_logic;
signal comp_PER_buf0, comp_PER_buf1 : std_logic;

begin
xors: for II in 15 downto 0 generate
  error(II) <= rdata(II) xor prbs(II);
  en(II) <= error(II) and count_en;
  pos_counter: counter_40bit port map (  
    clock => clk,  
    cnt_en => en_s(II),  
    sclr => sclr_s,  
    cout => cout(II),  
    q => q_count(II)  
  );
  holder: load_register port map (  
    clock => clk,  
    data => q_count(II),  
    enable => reg_en_s,  
    load => reg_load_s,  
    sclr => reg_sclr_s,  
    q => add_in(II)  
  );
end generate xors;

comp_PER_i <= error(0) or error(1) or error(2) or error(3) or error(4)  
or error(5) or error(6) or error(7) or error(8) or error(9)  
or error(10) or error(11) or error(12) or error(13) or  
error(14) or error(15);
comp_PER <= comp_PER_i;

total_error_adder : parallel_adder PORT MAP (  
  aclr => add_sclr,  
  clken => add_en,  
  clock => clk,
data0x => add_in(0),
data10x => add_in(10),
data11x => add_in(11),
data12x => add_in(12),
data13x => add_in(13),
data14x => add_in(14),
data15x => add_in(15),
data1x => add_in(1),
data2x => add_in(2),
data3x => add_in(3),
data4x => add_in(4),
data5x => add_in(5),
data6x => add_in(6),
data7x => add_in(7),
data8x => add_in(8),
data9x => add_in(9),
result => total_error
);

prbs_words_counter: counter_40bit PORT MAP (  
clock => clk,
cnt_en => count_words_s,
sclr => sclr_s,
cout => cout(16),
q => analyzed_prbs
);

cnt_en_s <= comp_PER_s and count_words_s;
PER_counter: counter_40bit port map (  
clock => clk,
cnt_en => cnt_en_s,
sclr => sclr_s,
cout => cout(17),
q => PER
);
simp_proc: process (clk)
  begin
    if rising_edge(clk) then
      en_s <= en;
      COUNTER_OVERFLOW <= cout(0) or cout(1) or cout(2) or cout(3)
      or cout(4) or cout(5) or cout(6) or cout(7)
      or cout(8) or cout(9) or cout(10) or cout(11) or cout(12) or cout(13) or cout(14) or cout(15) or cout(16) or cout(17);
      comp_PER_s <= comp_PER_i;
      count_words_s <= count_words;
      sclr_s <= sclr;
      reg_en_s <= reg_en;
      reg_load_s <= reg_load;
      reg_sclr_s <= reg_sclr;
      comp_PER_buf0 <= comp_PER_i;
      comp_PER_buf1 <= comp_PER_buf0;
      seq_err_flag <= comp_PER_buf0 and comp_PER_buf1;
      end if;
    end process;
end comp_behavioral;
entity FSM_control is
  port(
    clk : in std_logic;
    rst : in std_logic;
    rdata_rdempty : in std_logic;
    rdata_rdfull : in std_logic;
    rdata_aclr : out std_logic;
    rdata_rdreq : out std_logic;
    prbs_address : out std_logic_vector(16 downto 0);
    comp_analyzed_prbs : in std_logic_vector(39 downto 0);
    comp_PER : in std_logic;
    comp_sclr : out std_logic;
    comp_count_en : out std_logic;
    comp_count_words : out std_logic;
    reg_en : out std_logic;
    reg_load : out std_logic;
    reg_sclr : out std_logic;
    add_en : out std_logic;
    add_sclr : out std_logic;
    state_flag : out std_logic_vector(6 downto 0);
  );
end FSM_control;

architecture FSM_control_behav of FSM_control is

  type state_type is(
    RESET, IDLE, SYNCHRONIZATION, COMPARISON, SUM, SHOW, FULL);

  signal state, state_next : state_type := RESET;
  signal comp_sclr_i, comp_count_en_i : std_logic := '0';
  signal comp_count_words_i : std_logic := '0';
  signal rdata_aclr_i, rdata_rdreq_i : std_logic := '0';
  signal reg_en_i, reg_load_i, reg_sclr_i : std_logic := '0';
  signal add_en_i, add_sclr_i : std_logic := '0';
  signal cnt_sync, cnt_sync_i : natural range 0 to 17 := 0;
  signal cnt_sum, cnt_sum_i : natural range 0 to 33 := 0;
  signal prbs_address_i : natural range 0 to 131071 := 50;
  signal temp_address : natural range 0 to 131071 := 50;
  signal temp_address_out : natural range 0 to 131071 := 50;

begin
  temp_address <= temp_address_out;
  sync_proc : process(clk)
  begin
    if rising_edge(clk) then
      if rst = '1' then
        comp_sclr <= '1';
        comp_count_en <= '0';
        comp_count_words <= '0';
        reg_en <= '0';
      else
        state_next <= state;
        comp_sclr_i <= comp_sclr;
        comp_count_en_i <= comp_count_en;
        comp_count_words_i <= comp_count_words;
        reg_en_i <= reg_en;
        reg_load_i <= reg_load;
        reg_sclr_i <= reg_sclr;
        add_en_i <= add_en;
        add_sclr_i <= add_sclr;
        cnt_sync_i <= cnt_sync;
        cnt_sum_i <= cnt_sum;
        prbs_address_i <= prbs_address;
        temp_address_out <= temp_address;
      end if;
    end if;
  end process;
end FSM_control_behav;
reg_load <= '0';
reg_sclr <= '1';
add_en <= '0';
add_sclr <= '1';
prbs_address <= "0000000000000100";
rd_data_aclr <= '1';
rd_data_rdreq <= '0';
cnt_sync <= 0;
cnt_sum <= 0;
state <= RESET;

else
  comp_sclr <= comp_sclr_i;
  comp_count_en <= comp_count_en_i;
  comp_count_words <= comp_count_words_i;
  prbs_address <= std_logic_vector(to_unsigned(prbs_address_i,17));
  temp_address_out <= prbs_address_i;
  rd_data_aclr <= rd_data_aclr_i;
  rd_data_rdreq <= rd_data_rdreq_i;
  reg_en <= reg_en_i;
  reg_load <= reg_load_i;
  reg_sclr <= reg_sclr_i;
  add_en <= add_en_i;
  add_sclr <= add_sclr_i;
  cnt_sync <= cnt_sync_i;
  cnt_sum <= cnt_sum_i;
  state <= state_next;
end if;
end if;
end process;

--delta lambda process
state_decode_proc : process(
  state,
  rd_data_rdempty,
  rd_data_rdfull,
  temp_address,
  comp_PER,
  comp_analyzed_prbs,
  cnt_sync,
  cnt_sum )
begin
  state_next <= state;

  case (state) is
  when RESET =>
    state_flag <= "00000001";
    comp_sclr_i <= '1';
    comp_count_en_i <= '0';
    comp_count_words_i <= '0';
    reg_en_i <= '0';
    reg_load_i <= '0';
    reg_sclr_i <= '1';
    add_en_i <= '0';
    add_sclr_i <= '1';
    prbs_address_i <= 50;
    rd_data_aclr_i <= '1';
    rd_data_rdreq_i <= '0';
    cnt_sync_i <= 0;
Appendix A

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt_sum_i</td>
<td>&lt;= 0;</td>
</tr>
<tr>
<td>state_next</td>
<td>&lt;= IDLE;</td>
</tr>
</tbody>
</table>

**when IDLE =>**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_flag</td>
<td>&lt;= &quot;0000010&quot;;</td>
</tr>
<tr>
<td>comp_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>comp_count_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>comp_count_words_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_load_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>add_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>add_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address;</td>
</tr>
<tr>
<td>rdata_aclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>rdata_rdreq_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>cnt_sync_i</td>
<td>&lt;= 0;</td>
</tr>
<tr>
<td>cnt_sum_i</td>
<td>&lt;= 0;</td>
</tr>
</tbody>
</table>

**if** (rdata_rdempty = '1') **then**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_next</td>
<td>&lt;= IDLE;</td>
</tr>
</tbody>
</table>

**else**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_next</td>
<td>&lt;= SYNCHRONIZATION;</td>
</tr>
</tbody>
</table>

**end if;**

**when SYNCHRONIZATION =>**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_flag</td>
<td>&lt;= &quot;0000100&quot;;</td>
</tr>
<tr>
<td>comp_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>comp_count_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>comp_count_words_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_load_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>reg_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>add_en_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>add_sclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address;</td>
</tr>
<tr>
<td>rdata_aclr_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>rdata_rdreq_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>cnt_sync_i</td>
<td>&lt;= 0;</td>
</tr>
<tr>
<td>cnt_sum_i</td>
<td>&lt;= 0;</td>
</tr>
</tbody>
</table>

**if** (temp_address = 131070) **then**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>prbs_address_i</td>
<td>&lt;= 0;</td>
</tr>
</tbody>
</table>

**else**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address + 1;</td>
</tr>
</tbody>
</table>

**end if;**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt_sync_i</td>
<td>&lt;= cnt_sync + 1;</td>
</tr>
<tr>
<td>rdata_rdreq_i</td>
<td>&lt;= '0';</td>
</tr>
</tbody>
</table>

**if** (rdata_rdempty = '1') **then**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_next</td>
<td>&lt;= IDLE;</td>
</tr>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address;</td>
</tr>
<tr>
<td>cnt_sync_i</td>
<td>&lt;= cnt_sync;</td>
</tr>
</tbody>
</table>

**elsif** (rdata_rdfull = '1') **then**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>state_next</td>
<td>&lt;= FULL;</td>
</tr>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address;</td>
</tr>
<tr>
<td>cnt_sync_i</td>
<td>&lt;= cnt_sync;</td>
</tr>
</tbody>
</table>

**else**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
</table>
| if** (comp_PER = '0' and cnt_sync = 0) then**
| rdata_rdreq_i     | <= '0';               |
| state_next        | <= SYNCHRONIZATION;   |

**elsif** (cnt_sync = 1) **then**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdata_rdreq_i</td>
<td>&lt;= '0';</td>
</tr>
<tr>
<td>prbs_address_i</td>
<td>&lt;= temp_address;</td>
</tr>
<tr>
<td>state_next</td>
<td>&lt;= SYNCHRONIZATION;</td>
</tr>
</tbody>
</table>

**elsif** (cnt_sync = 2) **then**
else if (comp_PER = '0' and cnt_sync = 3) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (cnt_sync = 4) then
  rdata_rdreq_i <= '0';
  prbs_address_i <= temp_address;
  state_next <= SYNCHRONIZATION;
elsif (cnt_sync = 5) then
  rdata_rdreq_i <= '0';
  prbs_address_i <= temp_address;
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 6) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (cnt_sync = 7) then
  rdata_rdreq_i <= '0';
  prbs_address_i <= temp_address;
  state_next <= SYNCHRONIZATION;
elsif (cnt_sync = 8) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 9) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (cnt_sync = 10) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 11) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 12) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 13) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 14) then
  rdata_rdreq_i <= '1';
  state_next <= SYNCHRONIZATION;
elsif (comp_PER = '0' and cnt_sync = 15) then
  cnt_sync_i <= 0;
  rdata_rdreq_i <= '1';
  state_next <= COMPARISON;
else
  rdata_rdreq_i <= '1';
  cnt_sync_i <= 0;
  state_next <= SYNCHRONIZATION;
  prbs_address_i <= temp_address;
end if;
end if;

when COMPARISON =>
  state_flag <= "00010000";
  comp_sclr_i <= '0';
  comp_count_en_i <= '1';
  comp_count_words_i <= '1';
  reg_en_i <= '0';
  reg_load_i <= '0';
appendix

reg_sclr_i <= '0';
add_en_i <= '0';
add_sclr_i <= '0';
if (temp_address = 131070) then
  prbs_address_i <= 0;
else
  prbs_address_i <= temp_address + 1;
end if;
rdata_aclr_i <= '0';
rdata_rdreq_i <= '1';
cnt_sync_i <= 0;
cnt_sum_i <= 0;
if (rdata_rdempty = '1') then
  state_next <= IDLE;
elsif (rdata_rdfull = '1') then
  state_next <= FULL;
else
  --if (comp_analyzed_prbs = x"005F5E1000") then --40''
    if (comp_analyzed_prbs = x"7FFFFFFFFE") then --7H48'
      comp_count_en_i <= '0';
      comp_count_words_i <= '0';
      rdata_rdreq_i <= '0';
      reg_en_i <= '1';
      reg_load_i <= '1';
      state_next <= SUM;
    end if;
  end if;
when SUM =>
  state_flag <= "0010000";
  comp_sclr_i <= '0';
  comp_count_en_i <= '0';
  comp_count_words_i <= '0';
  reg_en_i <= '0';
  reg_load_i <= '0';
  reg_sclr_i <= '0';
  add_sclr_i <= '0';
  prbs_address_i <= 50;
  rdata_rdreq_i <= '0';
  rdata_aclr_i <= '0';
  cnt_sync_i <= 0;
if cnt_sum = 33 then
  add_en_i <= '0';
  cnt_sum_i <= 0;
  state_next <= SHOW;
else
  add_en_i <= '1';
  cnt_sum_i <= cnt_sum +1;
  state_next <= SUM;
end if;
when SHOW =>
  state_flag <= "0100000";
  comp_sclr_i <= '0';
  comp_count_en_i <= '0';
  comp_count_words_i <= '0';
  reg_en_i <= '0';
  reg_load_i <= '0';
  reg_sclr_i <= '0';
add_en_i <= '0';
add_sclr_i <= '0';
prbs_address_i <= 50;
rdata_aclr_i <= '0';
rdata_rdreq_i <= '0';
cnt_sync_i <= 0;
cnt_sum_i <= 0;

when FULL =>
  state_flag <= "1000000";
  comp_sclr_i <= '0';
  comp_count_en_i <= '0';
  comp_count_words_i <= '0';
  reg_en_i <= '0';
  reg_load_i <= '0';
  reg_sclr_i <= '0';
  add_en_i <= '0';
  add_sclr_i <= '1';
  prbs_address_i <= temp_address;
  rdata_aclr_i <= '1';
  rdata_rdreq_i <= '0';
  cnt_sync_i <= 0;
  cnt_sum_i <= 0;
  state_next <= IDLE;

end case;
end process;
end architecture;